JS_Beginning_to_Mastery_Part1_1

April 21, 2023

[ ]:
```
We can create variables using both `let` and `var` keywords.
```

In modern JavaScript, it's generally recommended to use `let` instead of `var` to declare variables.

The main reason for this is that `let` is block-scoped, whereas `var` is function-scoped. This means that a variable declared with `let` is only accessible within the block it was declared in (including any nested blocks), while a variable declared with `var` is accessible within the entire function it was declared in.

For example:

[13]:
```
function example() {
  var x = 1;
  if (true) {
    var x = 2; // same variable as above
    console.log(x); // 2
  }
  console.log(x); // 2
}

function example2() {
  let y = 1;
  if (true) {
    let y = 2; // different variable than above
    console.log(y); // 2
  }
  console.log(y); // 1
}
```

[14]:
```
example()
```

```
2
2
```

[15]:
```
example2()
```

```
2
1
```

In the first example, the variable `x` is overwritten within the nested block, and this change is reflected outside of the block as well. In the second example, the variable `y` is only accessible

within the block it was declared in, and a separate variable with the same name could be declared outside of the block without interfering with it.

In general, using `let` can help prevent accidental variable reassignment and make code easier to reason about, especially when dealing with more complex functions and control flow. However, there may be some cases where using var is still appropriate, such as when you intentionally want to create a variable that's accessible throughout an entire function

[1]:
```
console.log("Hello from file1.js");
console.log("Hello again");
```

```
Hello from file1.js
Hello again
```

ctrl + forward slash for comment

[5]:
```
// ctrl + forward slash for comment
// console.log();

let age = 20;
let firstName = "John";
```

```
evalmachine.<anonymous>:1
// ctrl + forward slash for comment
^

SyntaxError: Identifier 'age' has already been declared
    at Script.runInThisContext (node:vm:129:12)
    at Object.runInThisContext (node:vm:313:38)
    at run ([eval]:1020:15)
    at onRunRequest ([eval]:864:18)
    at onMessage ([eval]:828:13)
    at process.emit (node:events:513:28)
    at emit (node:internal/child_process:937:14)
    at process.processTicksAndRejections (node:internal/process/task_queues:83:
 ↪21)
```

[79]:
```
console.log(typeof(age)); // same as the below
console.log(typeof age);
```

```
number
number
```

0.0.1 convert number to string

[4]:
```
// convert number to string
age = String(age);
console.log(typeof age);
```

```
string
```

Convert number to string – again

[80]:
```
age = age.toString();
console.log(typeof age);
```

```
string
```

[81]:
```
console.log(age)
```

```
20
```

[82]:
```
// convert number to string -- again
age = age + "";
console.log(typeof age);
```

```
string
```

Convert string to number

[6]:
```
// convert string to number
age = Number(age);
console.log(typeof age);
```

```
number
```

[7]:
```
// convert string to number -- again
age = +age;
console.log(typeof age);
```

```
number
```

[8]:
```
// convert string to number -- again
age = parseInt(age);
console.log(typeof age);
```

```
number
```

String concatenation

[9]:
```
let string1 = "Shammu";
let string2 = "Khan";

let fullName = string1 + " " + string2;
console.log(fullName);
```

```
Shammu Khan
```

[10]:
```
// string concatenation -- again
let string3 = "17";
let string4 = "20";
```

```
let addedString = string3 + string4;
console.log(addedString);
```

```
1720
```

[11]:
```
// converts string to number and then adds
let addedNumber = +string3 + +string4; // +string3 converts string to number
console.log(addedNumber);
```

```
37
```

Template string

[12]:
```
// template string
let age2 = 22;
var firstName2 = "John";

console.log(`My name is ${firstName2} and I am ${age2} years old.`);
```

```
My name is John and I am 22 years old.
```

[2]:
```
// template string -- again
let aboutMe = `My name is ${firstName2} and I am ${age2} years old` ;
```

[3]:
```
console.log(aboutMe);
```

```
My name is John and I am 22 years old
```

[16]:
```
let myVariable = null;
console.log(myVariable); // prints null
myVariable = "ss";
console.log(myVariable, typeof myVariable); // ss string
console.log(typeof null); // this is bug or error as it shows object
```

```
null
ss string
object
```

BigInt

[17]:
```
// BigInt
let bigInt = 12345678901234567890123456789012345678901234567890n;
```

[18]:
```
console.log(bigInt, typeof bigInt);
```

```
12345678901234567890123456789012345678901234567890n bigint
```

[19]:
```
// BigInt -- again
let number = BigInt(123456);
console.log(number, typeof number);
```

```
123456n bigint
```

[20]:
```
console.log(Number.MAX_SAFE_INTEGER);
console.log(Number.MIN_SAFE_INTEGER);
```

```
9007199254740991
-9007199254740991
```

[21]:
```
// BigInt -- again
let number2 = 123n;
console.log(number2, typeof number2); // 123n bigint

console.log(number + number2); // 123579n
```

```
123n bigint
123579n
```

0.0.2 Booleans and comparison operators

[22]:
```
let a = 7;
let b = 8;
```

[23]:
```
console.log(a >= b);
```

```
false
```

== vs === == checks only value and === checks value and type

[24]:
```
// == vs ===
// == checks only value and === checks value and type
console.log(a == 2); // this is false as it checks only value
console.log(a === 2); // this is false as it checks value and type
console.log(a == "7"); // this is true as it checks only value
console.log(a === "7"); // this is false as it checks value and type
```

```
false
false
true
false
```

!= vs !== != checks only value and !== checks value and type

[25]:
```
console.log(a != 2);
console.log(a !== 2);
console.log(a != "7"); // this is false as it checks only value
console.log(a !== "7"); // this is true as it checks value and type
```

```
true
true
false
true
```

0.0.3 if conditional statement

[26]:
```
let age3 = 8;
let drink = age3 >= 18 ? "coffee" : "juice";
console.log(drink);
```

```
juice
```

[27]:
```
let firstName3 = "John";
let weight = 62;
let height = 1.8;

if(firstName3[0] === "J" && weight > 50 && height > 1.8){
    console.log("You are eligible to play");
} else {
    console.log("You are not eligible to play");
}
```

```
You are not eligible to play
```

[28]:
```
let weight2 = 62;
let height2 = 1.8;

let message = firstName3[0] === "J" || weight2 > 50 || height2 > 1.8 ? "You are␣
 ↪eligible to play" : "You are not eligible to play";
console.log(message); // You are eligible to play
```

```
You are eligible to play
```

0.0.4 switch statement

[29]:
```
let weight3 = 62;
let height3 = 1.8;

switch(firstName3[0]){
    case "J":
        console.log("You are eligible to play");
        break;
    case "A":
        console.log("You are eligible to play");
        break;
    case "B":
        console.log("You are eligible to play");
        break;
    default:
        console.log("You are not eligible to play");
}
```

```
You are eligible to play
```

[30]:
```
// groups multiple cases together
switch(firstName3[0]){
    case "J":
    case "A":
    case "B":
        console.log("You are eligible to play");
        break;
    default:
        console.log("You are not eligible to play");
}
```

```
You are eligible to play
```

[19]:
```
let winningNumber = 7;
//let userGuess = +prompt("Guess a number between 1 and 10"); // +prompt␣
 ↪converts string to number
let userGuess = 7;
```

[20]:
```
if(userGuess === winningNumber) {
  alert("You win!");
} else if(userGuess > winningNumber) {
  alert("Too high, try again!");
} else {
  alert("Too low, try again!");
}
```

```
evalmachine.<anonymous>:2
  alert("You win!");
  ^

ReferenceError: alert is not defined
    at evalmachine.<anonymous>:2:3
    at Script.runInThisContext (node:vm:129:12)
    at Object.runInThisContext (node:vm:313:38)
    at run ([eval]:1020:15)
    at onRunRequest ([eval]:864:18)
    at onMessage ([eval]:828:13)
    at process.emit (node:events:513:28)
    at emit (node:internal/child_process:937:14)
    at process.processTicksAndRejections (node:internal/process/task_queues:83:
 ↪21)
```

[51]:
```
let userGuess = 7;
```

```
evalmachine.<anonymous>:1
let userGuess = 7;
^
```

```
SyntaxError: Identifier 'userGuess' has already been declared
    at Script.runInThisContext (node:vm:129:12)
    at Object.runInThisContext (node:vm:313:38)
    at run ([eval]:1020:15)
    at onRunRequest ([eval]:864:18)
    at onMessage ([eval]:828:13)
    at process.emit (node:events:513:28)
    at emit (node:internal/child_process:937:14)
    at process.processTicksAndRejections (node:internal/process/task_queues:83:
↪21)
```

0.0.5 for loop

[31]:
```
for(let i=0; i <10; i++){
    console.log(i);
}
```

```
0
1
2
3
4
5
6
7
8
9
```

Sum of numbers between 0 and 10.

[32]:
```
let total = 0;
let num = 10;

for(let i=0; i <= num; i++){
    total += i;
}

console.log(total); // sum of numbers between 0 and 10
```

```
55
```

while loop

[33]:
```
// while loop

let i = 0;
while(i <= 10){
    console.log(i);
```

```
    i++;
}
```

```
0
1
2
3
4
5
6
7
8
9
10
```

[33]: 10

do while loop

[24]:
```
let j = 0;
do{
    console.log(j);
    j++
}while(j <=10);
```

```
0
1
2
3
4
5
6
7
8
9
10
```

[24]: 10

0.0.6 arrays

[34]:
```
// arrays
let fruits = ["apple", "banana", "orange", "mango", "grapes"];
let numbers = [10, 34, 56, 78, 90, 100];
let mixed = ["apple", 10, "banana", 34, "orange", 56, "mango", 78, "grapes",␣
 ↪90, 100];
```

[35]:
```
console.log(fruits); // ["apple", "banana", "orange", "mango", "grapes"]
fruits[1] = "pineapple";
```

```
console.log(fruits); // ["apple", "pineapple", "orange", "mango", "grapes"]
```

```
[ 'apple', 'banana', 'orange', 'mango', 'grapes' ]
[ 'apple', 'pineapple', 'orange', 'mango', 'grapes' ]
```

[36]:
```
console.log(numbers); // [10, 34, 56, 78, 90, 100]
numbers[1] = 100;
console.log(numbers); // [10, 100, 56, 78, 90, 100]
```

```
[ 10, 34, 56, 78, 90, 100 ]
[ 10, 100, 56, 78, 90, 100 ]
```

[37]:
```
console.log(mixed); // ["apple", 10, "banana", 34, "orange", 56, "mango", 78,␣
 ↪"grapes", 90, 100]
mixed[1] = 100;
console.log(mixed); // ["apple", 100, "banana", 34, "orange", 56, "mango", 78,␣
 ↪"grapes", 90, 100]
```

```
[
  'apple',  10,
  'banana', 34,
  'orange', 56,
  'mango',  78,
  'grapes', 90,
  100
]
[
  'apple',  100,
  'banana', 34,
  'orange', 56,
  'mango',  78,
  'grapes', 90,
  100
]
```

arrays are reference types and reference types are objects

[29]:
```
console.log(typeof fruits); // object
```

```
object
```

In JavaScript, an object literal is a way of creating a new object by specifying its properties and values in a concise syntax.

Here's an example of an object literal:

[38]:
```
const person = {
    name: 'John',
    age: 30,
    gender: 'male',
    occupation: 'developer'
```

```
};
```

In this example, we're using an object literal to create a new object called person. The object has four properties: name, age, gender, and occupation. The values of these properties are specified after the property name, separated by a colon. Each property-value pair is separated by a comma.

[39]:
```
console.log(person); // prints the object
```

```
{ name: 'John', age: 30, gender: 'male', occupation: 'developer' }
```

Object literals can also contain methods (i.e., functions that are properties of the object). Here's an example:

[40]:
```
const calculator = {
    add: function(x, y) {
      return x + y;
    },
    subtract: function(x, y) {
      return x - y;
    }
};
```

In this example, we're creating a new object called calculator with two methods: add and subtract. The methods are defined using function expressions and are assigned as properties of the object.

[41]:
```
// using dot notation to call the add and subtract methods
const sum = calculator.add(4, 9);
const difference = calculator.subtract(10, 4);
console.log(`The sum of 4 and 9 is ${sum}`);
console.log(`The difference between 10 and 4 is ${difference}`);
```

```
The sum of 4 and 9 is 13
The difference between 10 and 4 is 6
```

[42]:
```
let obj = {}; // object literal
console.log(typeof obj); // object
console.log(`type of obj is Array, true or false -- ${Array.isArray(obj)}`); //␣
 ↪true
```

```
object
type of obj is Array, true or false -- false
```

0.0.7 array methods

push() – adds element at the end

[43]:
```
// array methods
// push()
fruits.push("strawberry"); // adds element at the end
console.log(`push() adds strawberry at the end -- ${fruits}`);
```

```
push() adds strawberry at the end --
apple,pineapple,orange,mango,grapes,strawberry
```

pop() – removes last element

[44]:
```
// pop
fruits.pop(); // removes last element
console.log(`pop() removes last element or strawberry -- ${fruits}`);
```

```
pop() removes last element or strawberry -- apple,pineapple,orange,mango,grapes
```

unshift() – adds element at the beginning

[45]:
```
// unshift
fruits.unshift("strawberry"); // adds element at the beginning
console.log(`unshift(strawberry) adds strawberry at the beginning --␣
 ↪${fruits}`);
```

```
unshift(strawberry) adds strawberry at the beginning --
strawberry,apple,pineapple,orange,mango,grapes
```

shift() – removes first element

[47]:
```
// shift
fruits.shift(); // removes first element
console.log(`shift() removes first element or removes strawberry -- ${fruits}`);
```

```
shift() removes first element or removes strawberry --
pineapple,orange,mango,grapes
```

splice() – removes elements from array

[48]:
```
// splice -- removes elements from array
fruits.splice(1, 2); // removes 2 elements from index 1
console.log(`splice(1, 2) removes 2 elements from index 1 and so removes␣
 ↪pineapple and orange -- ${fruits}`);
```

```
splice(1, 2) removes 2 elements from index 1 and so removes pineapple and orange
-- pineapple,grapes
```

splice() – adds elements to array

[49]:
```
// exact opposite of the previous one
fruits.splice(1, 0, "banana", "mango"); // adds 2 elements at index 1
console.log(`fruits.splice(1, 0, "banana", "mango") adds banana and mango at␣
 ↪index 1 -- ${fruits}`);
```

```
fruits.splice(1, 0, "banana", "mango") adds banana and mango at index 1 --
pineapple,banana,mango,grapes
```

concat()

[50]:
```
let vegetables = ["tomato", "potato", "brinjal"];
let all = fruits.concat(vegetables);
```

```
console.log(all);
```

```
[
  'pineapple',
  'banana',
  'mango',
  'grapes',
  'tomato',
  'potato',
  'brinjal'
]
```

slice() – returns elements from one index to another

[51]:
```
let sliced = all.slice(1, 4); // returns elements from index 1 to 3
console.log(`slice(1, 4) returns elements from index 1 to 3 -- ${sliced}`);
```

```
slice(1, 4) returns elements from index 1 to 3 -- banana,mango,grapes
```

reverse() – reverses the array

[52]:
```
// reverse
console.log(`reverse() reverses the array -- ${all.reverse()}`); // reverses␣
 ↪the array
```

```
reverse() reverses the array --
brinjal,potato,tomato,grapes,mango,banana,pineapple
```

sort() – sorts an array

[53]:
```
console.log(`sort() sorts the array -- ${all.sort()}`); // sorts the array
```

```
sort() sorts the array -- banana,brinjal,grapes,mango,pineapple,potato,tomato
```

join() – joins the elements of an array by a symbol provided in the function

[54]:
```
console.log(`join(' - ') joins the array elements by ' - ':   ${all.join(' -␣
 ↪')}`); // joins the array elements with -
```

```
join(' - ') joins the array elements by ' - ':   banana - brinjal - grapes -
mango - pineapple - potato - tomato
```

indexOf() – returns index of the element

[55]:
```
console.log(`all array -- ${all}`);
console.log(`indexOf() gives the index of "banana" in all -- ${all.
 ↪indexOf("banana")}`); // returns index of the element
```

```
all array -- banana,brinjal,grapes,mango,pineapple,potato,tomato
indexOf() gives the index of "banana" in all -- 0
```

lastIndexOf() – returns the index of the last occurrence of the specified value

[56]:
```
console.log(`lastIndexOf() gives the index of the last occurrence of "banana"␣
↪${all.lastIndexOf("banana")}`); // returns index of the last occurrence of␣
↪the element "banana"
```

```
lastIndexOf() gives the index of the last occurrence of "banana" 0
```

includes() – returns true if an element is present in the array

[57]:
```
console.log(all.includes("banana")); // returns true if element is present
```

```
true
```

0.0.8 find method

returns the first element of an array that satisfies some condition.

[58]:
```
let numbers2 = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100];
let found = numbers2.find(function(value){
    return value > 50;
});
console.log(found); // returns first element greater than 50
```

```
60
```

0.0.9 findIndex()

returns the index of the first element that satisfies some condition.

[59]:
```
let found2 = numbers2.findIndex(function(value){
    return value > 50;
});
console.log(found2); // returns index of first element greater than 50
```

```
5
```

0.0.10 filter()

returns all the elements that satisfies some condition.

[60]:
```
let filtered = numbers2.filter(function(value){
    return value > 50;
});
console.log(filtered); // returns all elements greater than 50
```

```
[ 60, 70, 80, 90, 100 ]
```

0.0.11 forEach()

Gives both value and index which allows looping through an array

[61]:
```
numbers2.forEach(function(value, index){
    console.log(value, index);
```

```
});
```

```
10 0
20 1
30 2
40 3
50 4
60 5
70 6
80 7
90 8
100 9
```

[62]:
```
let array5 = ["item1", "item2"];
let array6 = array5;

console.log(array5);
console.log(array6);
```

```
[ 'item1', 'item2' ]
[ 'item1', 'item2' ]
```

[63]:
```
array5.push("item3");
console.log(`After pushing element to array5, array5 is ${array5}`);
```

```
After pushing element to array5, array5 is item1,item2,item3
```

[65]:
```
console.log(`Now, array6 is also changed to: ${array6}`)
```

```
Now, array6 is also changed to: item1,item2,item3
```

0.0.12 clone array

[1]:
```
let array10 = ["item1", "item2"];
```

Clone – First way

Use three dots ...

[2]:
```
console.log("First method using ...");
let array11 = [...array10];
```

```
First method using …
```

[3]:
```
console.log(array11);
```

```
[ 'item1', 'item2' ]
```

[4]:
```
array10.push("item3"); // this will have no effect on array11, a clone of␣
↪array10
console.log("After pushing element to array10");
```

```
// Now, array10 and array11 are different
console.log("Are array10 and array11 are the same", array10 === array11);
```

```
After pushing element to array10
Are array10 and array11 are the same false
```

[5]:
```
console.log(array11) // "item3" is not added to array11
```

```
[ 'item1', 'item2' ]
```

Clone – Second way

slice()

[6]:
```
console.log("Second method using slice");
let array12 = array10.slice();
console.log(array12);
```

```
Second method using slice
[ 'item1', 'item2', 'item3' ]
```

[7]:
```
array10.push("item4");
console.log("After pushing element to array10");
console.log("Are array10 and array12 are the same", array10 === array12);
```

```
After pushing element to array10
Are array10 and array12 are the same false
```

[8]:
```
console.log("array10",array10);
console.log("array12",array12);
console.log("");
```

```
array10 [ 'item1', 'item2', 'item3', 'item4' ]
array12 [ 'item1', 'item2', 'item3' ]
```

Clone – Third way

concat()

[9]:
```
console.log("Third method using concat");
let array13 = array10.concat();
```

```
Third method using concat
```

[11]:
```
array10.push("item5");
console.log(`After pushing element to array10, array10 is ${array10}`);
console.log("Are array10 and array13 are the same", array10 === array13);
```

```
After pushing element to array10, array10 is item1,item2,item3,item4,item5,item5
Are array10 and array13 are the same false
```

[14]:
```
console.log("array10",array10);
console.log("array13",array13);
```

```
array10 [ 'item1', 'item2', 'item3', 'item4', 'item5' ]
array13 [ 'item1', 'item2', 'item3', 'item4' ]
```

0.0.13 for loop

[15]:
```
let fruits3 = ["apple", "orange", "banana"];

let fruits4 = [];
for(let i=0; i<fruits3.length; i++) {
    fruits4.push(fruits3[i].toUpperCase());
}
console.log(fruits4);
```

```
[ 'APPLE', 'ORANGE', 'BANANA' ]
```

The value of `const` object can't be changed once the value is assigned.

[16]:
```
const pi=3.14;
// the below line will throw error
// pi= 12;
console.log(pi);
```

```
3.14
```

We can push to constant array. This will be stored in heap memory when doing push, we are not changing address and so we can push.

[17]:
```
const fruits5 = ["apple", "orange", "banana"];
fruits5.push("banana");
// the below line will throw error
// fruits5 = ["banana", "mango"]
```

[17]:
```
4
```

[18]:
```
console.log(fruits5)
```

```
[ 'apple', 'orange', 'banana', 'banana' ]
```

We can change the value of constant object. This will be stored in heap memory when doing push, we are not changing address and so we can push.

[19]:
```
const person10 = {
    name: "John",
    age: 30
};
person10.name = "Peter";
person10.age = 40;
// the below line will throw error
```

```
/*
person10 = {
     name: "Peter",
     age: 40
}
*/
```

[19]: 40

[20]:
```
console.log(person10)
```

```
{ name: 'Peter', age: 40 }
```

0.0.14 More Looping

let x of array

[21]:
```
const fruits10 = ["apple", "orange", "banana"];

// print all the fruits separately
for(let fruit of fruits10) {
    console.log(fruit);
}
```

```
apple
orange
banana
```

while loop

[22]:
```
// prints all the fruits in upper case
let k=0;
while(k<fruits10.length) {
    console.log(fruits10[k].toUpperCase());
    k++;
}
```

```
APPLE
ORANGE
BANANA
```

[22]: 2

0.0.15 array destructuring

[23]:
```
// saving array values to variables
const myArray = ["value1", "value2"];
let myVar1 = myArray[0];
let myVar2 = myArray[1];
console.log("value of myVar1", myVar1);
console.log("value of myVar2", myVar2);
```

```
value of myVar1 value1
value of myVar2 value2
```

[24]:
```
// array destructuring -- alternative to the previous steps
const myArray2 = ["value1", "value2"];
// the below line assigns the first value of the array to var1 and the second␣
 ↪value to var2
let [var1, var2] = myArray2; // this is called array destructuring
```

The above line assigns the first value of the array to `var1` and the second value to `var2`.

[3]:
```
var1 = "new value"; // we can do this as we are using let
console.log("value of var1", var1);
console.log("value of var2", var2);
```

```
value of var1 new value
value of var2 value2
```

But if we array destructure using const, we will not be able to change the values of the variables.

[25]:
```
const [constVar1, constVar2] = myArray2;
// constVar1 = "new value"; // this will throw an error
```

array destructuring – assigning values to less number of variables

[26]:
```
const myArray3 = ["value1", "value2", "value3"];
let [var3, var4] = myArray3;
console.log("value of var3", var3);
console.log("value of var4", var4);
```

```
value of var3 value1
value of var4 value2
```

array destructuring – assigning values to more number of variables

[27]:
```
const myArray4 = ["value1", "value2"];
let [var5, var6, var7] = myArray4;
console.log("value of var5", var5);
console.log("value of var6", var6);
console.log("value of var7", var7); // this will be undefined
```

```
value of var5 value1
value of var6 value2
value of var7 undefined
```

array destructuring – skipping values

[28]:
```
// skipping values
const myArray5 = ["value1", "value2", "value3"];
let [var8, , var9] = myArray5;
console.log("value of var8", var8);
console.log("value of var9", var9);
```

```
value of var8 value1
value of var9 value3
```

array destructuring – using rest operator ..

rest operator ..

[29]:
```
const myArray6 = ["value1", "value2", "value3", "value4"];
```

The below line assigns the first value of the array to var10 and the rest of the values to var11

[30]:
```
// the below line assigns the first value of the array to var10
// and the rest of the values to var11
let [var10, ...var11] = myArray6; // this is called rest operator
console.log("value of var10", var10);
console.log("value of var11", var11);
```

```
value of var10 value1
value of var11 [ 'value2', 'value3', 'value4' ]
```

[31]:
```
let myNewArray2 = myArray6.slice(2); // this assigns values from the third␣
 ↪index to the end of the array to myNewArray
console.log("value of myNewArray", myNewArray2);
```

```
value of myNewArray [ 'value3', 'value4' ]
```

array destructuring – more Can't do the following as rest operator should be the last element

[32]:
```
// let [..., var12] = myArray6; // can't do this as rest operator should be the␣
 ↪last element
// console.log(var12);
```

0.0.16 Objects

In JavaScript, objects are one of the fundamental data types, used to store and manipulate collections of data in a structured way.

An object in JavaScript is a container for properties, which are essentially key-value pairs. The keys are always strings, while the values can be any data type, including other objects, functions, arrays, and primitive types such as numbers and strings.

Objects can be created using object literals, which use curly braces {} to enclose the properties and their values:

[33]:
```
const person = {
  name: "John",
  age: 30,
  hobbies: ["reading", "playing guitar"],
  address: {
    street: "123 Main St",
    city: "Anytown",
```

```
    state: "CA"
  }
};
```

In this example, `person` is an object with four properties: `name`, `age`, `hobbies`, and `address`. The address property is itself an object with its own properties.

You can access the properties of an object using dot notation or square bracket notation:

[34]:
```
console.log(person.name);       // "John"
console.log(person["age"]);   // 30
console.log(person.hobbies[0]);   // "reading"
console.log(person.address.state); // "CA"
```

```
John
30
reading
CA
```

Objects in JavaScript are dynamic, which means you can add or remove properties at any time:

[35]:
```
person.job = "programmer";  // add a new property
delete person.hobbies;     // remove the hobbies property
```

[35]:
```
true
```

[36]:
```
console.log(person)
```

```
{
  name: 'John',
  age: 30,
  address: { street: '123 Main St', city: 'Anytown', state: 'CA' },
  job: 'programmer'
}
```

Objects can also have methods, which are functions stored as properties:

[37]:
```
const calculator = {
  add: function(a, b) {
    return a + b;
  },
  subtract: function(a, b) {
    return a - b;
  }
};

console.log(calculator.add(2, 3));       // 5
console.log(calculator.subtract(5, 2));  // 3
```

```
5
3
```

In this example, calculator is an object with two methods, add and subtract

[39]:
```
const person2 = {"name": "John", "age": 30, "city": "New York"};
```

0.0.17 Accessing object values using dot notation

[40]:
```
// accessing object values using dot notation
console.log(person.name);
console.log(person.age);
console.log(person.city);
```

```
John
30
undefined
```

[41]:
```
console.log(typeof person);
```

```
object
```

0.0.18 Accessing object values using bracket notation

[42]:
```
// accessing object values using bracket notation
console.log(person["name"]);
console.log(person["city"]);
```

```
John
undefined
```

0.0.19 We can have array as values of an object

[45]:
```
// we can have array as values of an object
const person4 = {"name": "John", "age": 30, "city": "New York", "hobbies":␣
↪["music", "movies", "sports"]};
console.log(person4.hobbies[1]);
```

```
movies
```

0.0.20 How to add key value pairs to an object

[47]:
```
// how to add key value pairs to an object
person4["email"] = "sha_is13@gmail.com";
console.log(person4);
```

```
{
  name: 'John',
  age: 30,
  city: 'New York',
  hobbies: [ 'music', 'movies', 'sports' ],
  email: 'sha_is13@gmail.com'
}
```

[48]:
```
console.log(person4.email);
console.log(person4["email"]);
```

```
sha_is13@gmail.com
sha_is13@gmail.com
```

0.0.21 Create new property

[49]:
```
const person6 = {
    "name": "John",
    "age": 30,
    "city": "New York",
    "hobbies": ["music", "movies", "sports"]
};

person6["gender"] = "male";
console.log(person6.gender);
```

```
male
```

0.0.22 Difference between dot and bracket notation

- dot notation is used when we know the key
- bracket notation is used when we don't know the key
- bracket notation is used when we have a variable as key
- bracket notation is used when we have a key with spaces

[50]:
```
const key = "email";
const person7 = {
    name: "John",
    age: 30,
    city: "New York",
    "person hobbies": ["music", "movies", "sports"]
};
```

[51]:
```
console.log(person7["person hobbies"]);
```

```
[ 'music', 'movies', 'sports' ]
```

Suppose we want the value of the variable key to be the key for person4 object. Now, using dot notation will not work.

[52]:
```
// Suppose we want the value of the variable key to be the key
// for person4 object. Now, using dot notation will not work

person4.key = "sha_is13@gmail.com";
console.log(person4);
```

```
{
  name: 'John',
```

```
  age: 30,
  city: 'New York',
  hobbies: [ 'music', 'movies', 'sports' ],
  email: 'sha_is13@gmail.com',
  key: 'sha_is13@gmail.com'
}
```

But we want the value of the variable key to be the key solution is using bracket notation

[53]:
```
// But we want the value of the variable key to be the key
// solution is using bracket notation
person4[key] = "sha_is13@gmaill.com";
console.log(person4);
```

```
{
  name: 'John',
  age: 30,
  city: 'New York',
  hobbies: [ 'music', 'movies', 'sports' ],
  email: 'sha_is13@gmaill.com',
  key: 'sha_is13@gmail.com'
}
```

0.0.23 Remove a key value pair from an object

[55]:
```
// remove a key value pair from an object
delete person4["key"];
console.log(person4);
```

```
{
  name: 'John',
  age: 30,
  city: 'New York',
  hobbies: [ 'music', 'movies', 'sports' ],
  email: 'sha_is13@gmaill.com'
}
```

0.0.24 Iterate objects

[57]:
```
const person8 = {
    name: "John",
    age: 30,
    city: "New York",
    hobbies: ["music", "movies", "sports"]
};
```

[62]:
```
// for in loop
for(let key in person8){
    console.log(key, person8[key]);
```

```
}
```

```
name John
age 30
city New York
hobbies [ 'music', 'movies', 'sports' ]
```

[63]:
```
// Object.keys() returns an array of keys
console.log(typeof (Object.keys(person8)));
```

```
object
```

[64]:
```
// for in loop -- another example
for(let key in person8){
    console.log(`${key}: ${person8[key]}`);
}
```

```
name: John
age: 30
city: New York
hobbies: music,movies,sports
```

[66]:
```
// for key of loop
for(let key of Object.keys(person8)){
    console.log(person8[key]);
}
```

```
John
30
New York
[ 'music', 'movies', 'sports' ]
```

0.0.25 Computed properties

[67]:
```
const key1 = "objkey1";
const key2 = "objkey2";

const value1 = "objvalue1";
const value2 = "objvalue2";

obj = {};
obj[key1] = value1;
obj[key2] = value2;
console.log(obj);
```

```
{ objkey1: 'objvalue1', objkey2: 'objvalue2' }
```

0.0.26 Spread operator ...

Spread operator is used to split up array elements or object properties

[69]:
```
const array1 = [1,2,3];
const array2 = [5,6,7];

const newArray = [...array1]; // copies array1 into newArray
console.log(newArray);
```

```
[ 1, 2, 3 ]
```

[70]:
```
const newArray2 = [...array1, ...array2]; // copies array1 and array2 into␣
 ↪newArray2
console.log(newArray2);
```

```
[ 1, 2, 3, 5, 6, 7 ]
```

[71]:
```
// spread operator -- continued
// for array2 we are not using spread operator
const newArray3 = [...array1, array2]; // copies values of array1 as elements␣
 ↪and array2 as array into newArray3
console.log(newArray3);
```

```
[ 1, 2, 3, [ 5, 6, 7 ] ]
```

[72]:
```
// spread operator -- continued
const newArray4 = [...array1, ...array2, 78, 56];
console.log(newArray4);
```

```
[
  1, 2,  3,  5,
  6, 7, 78, 56
]
```

[73]:
```
// spread string
const newArray5 = [..."Hello World"];
console.log(newArray5);
```

```
[
  'H', 'e', 'l', 'l',
  'o', ' ', 'W', 'o',
  'r', 'l', 'd'
]
```

Spread operator doesn't work for integers

[75]:
```
// spread operator in objects
const person0 = {
    key1: "value1",
    key2: "value2"
};
console.log(person0);
```

```
{ key1: 'value1', key2: 'value2' }
```

[77]:
```
// just to make things clear that same key can't
// be present more than one time in an object
const person22 = {
    key1: "value1",
    key2: "value2",
    key1: "LastkeyValue" // this will overwrite the previous key1
};
console.log(person22);
```

```
{ key1: 'LastkeyValue', key2: 'value2' }
```

[78]:
```
// spread operator in objects -- continued
const obj1 = {
    key1: "value1",
    key2: "value2"
};

const obj2 = {
    key3: "value3",
    key4: "value4"
};
```

[79]:
```
const newObject = { ...obj1}; // copies obj1 into newObject
console.log(newObject);
```

```
{ key1: 'value1', key2: 'value2' }
```

[80]:
```
// spread operator in objects -- continued
const newObject2 = { ...obj1, ...obj2}; // copies obj1 and obj2 into newObject2
console.log(newObject2);
```

```
{ key1: 'value1', key2: 'value2', key3: 'value3', key4: 'value4' }
```

suppose we are using spread operator to copy obj1 and obj2 and they both have a same key. In this case, the value of the key in obj2 will overwrite the value of the key in obj1.

[81]:
```
const obj3 = {
    key1: "value1",
    key2: "value2"
};

const obj4 = {
    key1: "value3",
    key4: "value4",
    key5: "value5"
};

const newObject3 = {...obj3, ...obj4}; // copies obj3 and obj4 into newObject3
console.log(newObject3);
```

```
{ key1: 'value3', key2: 'value2', key4: 'value4', key5: 'value5' }
```

[82]:
```
// spread operator in objects -- continued
// adding new key with spread operator
const newObject4 = {...obj3, ...obj4, key6: "value6"};
console.log(newObject4);
```

```
{
  key1: 'value3',
  key2: 'value2',
  key4: 'value4',
  key5: 'value5',
  key6: 'value6'
}
```

Spread string will result into index as keys and characters as values

[83]:
```
const newObject5 = {..."Hello World"};
console.log(newObject5);
```

```
{
  '0': 'H',
  '1': 'e',
  '2': 'l',
  '3': 'l',
  '4': 'o',
  '5': ' ',
  '6': 'W',
  '7': 'o',
  '8': 'r',
  '9': 'l',
  '10': 'd'
}
```

Spread array will result into index as keys and elements as values

[84]:
```
const newObjects = {...["item1", "item2"]};
console.log(newObjects);
```

```
{ '0': 'item1', '1': 'item2' }
```

[85]:
```
// spread string of characters
const newObjects2 = {..."abcdefghijklmnopqrstuvwxyz"};
console.log(newObjects2);
```

```
{
  '0': 'a',
  '1': 'b',
  '2': 'c',
  '3': 'd',
  '4': 'e',
```

```
    '5': 'f',
    '6': 'g',
    '7': 'h',
    '8': 'i',
    '9': 'j',
    '10': 'k',
    '11': 'l',
    '12': 'm',
    '13': 'n',
    '14': 'o',
    '15': 'p',
    '16': 'q',
    '17': 'r',
    '18': 's',
    '19': 't',
    '20': 'u',
    '21': 'v',
    '22': 'w',
    '23': 'x',
    '24': 'y',
    '25': 'z'
}
```

[ ]: