2 is third -- in a string, 1 is the second,

javascript string lastIndexOf():-

returns the index of last occurence of a specified text in a string.

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
let text="please locate where 'locate' occurs!";
let index = text.lastIndexOf("locate");
document.getElementById("demo").innerHTML
    =index;
</script>
</body>
</html>
```

=> 21

Both indexOf(), and lastIndexOf() return -1 if the text is not found:-

```
<!DOCTYPE html>
<html>
<body>
<p id ="demo" ></p>
<script>
let text="please locate where 'locate' occurs!";
let index=text.indexOf("John");
document.getElementById("demo").innerHTML
    =index;
</script>
</body>
</html>
```

=> -1

→ Both method accepts a second parameter as the starting position for the search:-

occurs!";

let index = text.indexOf("locate", 15);

→ The lastIndexOf() methods searches backwards (from the end of beginning, if the second parameter is 15, the search starts at a position 15 and searches to the beginning of string.

let text = "please locate where 'locate' occurs!";

text.lastIndexOf("locate", 15);

String search():

search() method searches a string for a string, and returns the position of the match

→ let text = "please locate where 'locate' occurs!";

text.search("locate");

→ let text = "please locate where 'locate' occurs!";

text.search(/locate/);

→ The two methods indexOf() and () search accept the same arguments and return same value.

→ The search() method cannot take a second start position argument.

→ The indexOf() method cannot take powerful search values (regular expressions)

Javascript String Match():

The match() method returns an array containing the results of matching a string against (or a regular expression).

perform a search for "ain"

let text = "The rain in SPAIN stays mainly in the plain";

text.match("ain");

perform a search for "ain":-

```
let text="the rain in SPAIN stays
        mainly in the plain";
text.match(/ain/);
```

Perform a global search for "ain":-

```
let text="the rain in SPAIN
```

Javascript String MatchAll()

→ The matchAll() method returns an iterator containing the results of matching a string against a string (or a regular expression).

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
  let text="I love cats. Cats are very
  easy to love. Cats are very popular".
Const iterator= text.matchAll("cats");
document.getElementById("demo").innerHTML.
  =Array.from(iterator);
</script>
</body>                    ⇒cats, cats.
</html>
```

→ If the parameter is a regular expression, the global flag (g) must be set, otherwise a TypeError is thrown.

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
let text="I love cats. Cats are very easy
  to love. Cats are very popular".
Const iterator= text.matchAll(/cats/g);
document.getElementById("demo").innerHTML
  =Array.from(iterator);
```

</body>
</html>

If you want to search case insensitive, the insensitive flag(i) must

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
let text = "I love cats. Cats are very
Easy to love. Cats are very popular".
const iterator = text.matchAll(/cats/gi);
document.getElementById("demo").innerHTML
= Array.from(iterator);
</script>
</body>
</html>
```

Javascript ~~[illegible]~~ string includes:-

→ The includes() method returns true if a string contains a specified value.

→ otherwise it returns false.

→ let text = "Hello world, welcome to the universe.";
text.includes("world");

→ check if a string includes "world" start at position 12.

```
let text = "Hello world, welcome to
the universe.";
text.includes("world", 12);
```

Javascript string startsWith():-

The startsWith() method returns true if a

text.startswith("Hello");

→ let text = "Hello world, welcome to universe!";
text.startswith("world")

→ A start position for the search can be specified.

let text = "Hello world, welcome to universe.";
text.startswith("world", 5)

→ let text = "Hello world, welcome to universe.";
text.startswith("world", 6)

Javascript string endswith():-

the endswith() method returns true if a string ends with a specified value.
→ otherwise it returns false:-

let text = "John Doe";
text.endswith("Doe");

→ check if the 11 first characters of a string ends with "world".

let text = "Hello world, welcome to universe";
text.endswith("world", 11);

Javascript template:-

Template Literals
Template strings
String templates
Back-tics syntax

Back-tics syntax:-

Template literals use backs-ticks (``) rather than the quotes ("") to define a string:-

<!DOCTYPE html>
<html>

```
<scipts>
let text= 'Hello World!';
document.getElement By Id("demo").i
    nne>HTML= text;
</scipts>
</body>
</html>
```

⇒ Hello world.

Quotes Inside strings:-

With template literals, you can use both single and double quotes inside a string.

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
let text= "He's often called 'Johnny'";
document.get ElementById("demo").innerHT
    ML = text;
</script>
</body>
</html>
```

⇒ He's often called 'Johnny'

Multiline strings:-

Template literals allows multiline strings:-

```
let text= 'The quick brown box
    jumps over the lazy dog';
```

Interpolation:-

→ Template literals provide an easy way to interpolate variables and expressions into

Variable substitutions:-

Template literals allow variables in strings:-

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
let firstName = "John";
let lastName = "Doe";
let text = `Welcome ${firstName}, ${lastName}!`;
document.getElementById("demo").innerHTML
    = text;
</script>
</html>
```

⇒ Welcome John, Doe.

→ Automatic replacing of variables with real values is called string interpolation.

Expression substitution:-

```
let price = 10;
let VAT = 0.25;
let total = `total: ${(price * (1+VAT)).toFixed
    (2)}`;
```

→ Automatic replacing of expressions with real values is called string interpolation.

HTML templates:-

```
let header = "Templates Literals";
let tags = ["template literals", "JavaScript",
    "ES6"];
let html = `<h2>${header}</h2><UL>`;
for (const x of tags) {
```

Javascript has only one type of numbers.
Numbers can be written with or without decimals.

```
let x = 3.14;
let y = 3;
```

```
<!doctype html>
<html>
<body>
<h2> Javascript Numbers </h2>
<p id="demo"></p>
<script>
let x = 123e5;
let y = 123e-5;
document.getElementById("demo").innerHTML = x + "<br>" + y;
</script>
</body>
</html>
```

Javascript Numbers are always 64-bit Floating point:-

→ Javascript numbers are always stored as double precision floating point numbers, following the international IEEE 754 standard.
→ This format stores numbers in 64 bits, where the number (the fraction) is stored in bits 0 to 51, the exponent in bits 52 to 62, and the sign in bit 63.

Value (Fraction) Exponent sign

let y = 999999999999999; o/p:-

let y = 9999999999999999;

will be 10000000000000000

→ The maximum no. number decimal decimals is 17.

floating precision:-

floating point arithematic is not always 100% accurate

let x = 0.2+0.1;

→ to solve the problem above, it helps to multiply and divide.

let x = (0.2 * 10 + 0.1 * 10) / 10;

Adding Numbers and strings:-

JavaScipt uses the + operator for both addition and concatenation.

→ Numbers are added. strings are concatenated.

→ If you add 2-numbers, the result will be a number;

let x = 10;

let y = 20;

let z = x + y;

→ If you add 2-strings, the result will be a string concatenation.

let x = "10";

let y = "20";

let z = x + y;

→ If you add a number and a string the result will be a string concatenation.

let x = 10;

let y = "20";

let z = x+y;

→ A common mistake to expect is 30

→ let x=10;
let y=20;
let z="The result is 30"+x+y;

o/p:- 1020

→ let x=20;
let y=30;
let z=10;
let result= x+y+z;

3030

→ The javascipt interpreter works from left to right.

→ First 10+20 is added because x and y are both numbers

Numeric strings:-

Javascipt strings can have numeric content.

let x=100;
let y="100";

→ Javascipt will try to convert strings to numbers in all numeric operations

let x="100";
let y="10";
let z= x/y;

o/p:- 10

→ let x="100";
let y="10";
let z=x*y;

o/p:- 1000

→ let x="100";
let y="10";

```
let x = "100";
let y = "10";
let z = x + y;
```

10006

NaN - Not a number:-

→ NaN is a JavaScript reserved word indicating that a number is not a legal number.

→ trying to do arithemetic with a non-numeric string will result in NaN (Not a Number):-

```
let x = 100 / "Apple";
```

↓

→ ~~However~~ NaN

→ However, if the string numeric, the result will be a number:-

```
let x = 100 / "10";
```

→ 10

you can use the global JavaScript function isNaN() to find out if a value is not a number.

```
let x = 100 / "Apple";
isNaN(x);
```

⇒ true

→ Watch out for NaN. If you use NaN in a mathematical operation, the result will be NaN:-

```
let x = NaN;
let y = 5;        ⇒ NaN
let z = x + y;
```

NaN is a number : type of NaN returns number:-

```
typeof NaN;
```

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
let x = NaN;
```

document.getElementBy
=typeof x;
</script>
</body>
</html>

→ the typeof NaN is number
number.

Infinity :- Infinity or (-Infinity)

Infinity (or -Infinity) is the value JavaScript will return if you calculate a number outside the largest possible number.

```
let myNumber=2;
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
<script>
let myNumber=2;
let txt="";
while (myNumber !=Infinity){
myNumber=myNumber*myNumber;
txt=txt+myNumber+"<br>";
}
document.getElementById("demo").innerHTML
=txt;
</script>
</html>
</body>
```

4
16
256
65536
4294967296
18446744073709551616
.
.
.
until Infinity.