# ES6 Rest Parameters

- Prior to ES6, if we wanted to pass a variable number of arguments to a function, we had to use the `arguments` object, an array like object to access parameters passed to a function.

```
// for example lets write a function that array contains all the arguments
passed to it
function containsAll(arr) {
  for (let k = 1; k < arguments.length; k++) {
    let num = arguments[k];
    if (arr.indexOf(num) === -1) {
      return false;
    }
  }
  return true;
}
let x = [1, 2, 3, 4, 5];
console.log(containsAll(x, 1, 2, 3)); // true
console.log(containsAll(x, 1, 2, 3, 4)); // false
```

- In ES6, we can use the `...` syntax to pass a variable number of arguments to a function.

```
function containsAll(arr, ...nums) {
  for (let num of nums) {
    if (arr.indexOf(num) === -1) {
      return false;
    }
  }
  return true;
}
```

- The ...nums parameter here is called the rest parameter. It is an array-like object that contains all the arguments passed to the function after the first parameter.
- The ... are called the Spread Operator.

## Note

- Only the last parameter can be a rest parameter. If there are no extra arguments, the rest parameter will simply be an empty array, the rest parameter will never be undefined.
- The rest parameter can only be used in function calls.

## Quick Quiz

- What will be the output of the following code?

```
function magic(...nums) {
  let sum = 0;
  nums.filter((n) => n % 2 == 0).map((el) => (sum += el));
  return sum;
}
console.log(magic(1, 2, 3, 4, 5, 6));
```

- 12

## Spread Operator

- The operator `...` is used to spread an array into a list of arguments.
- It is simmilar to the rest parameter, but it is used to spread an array into a list of arguments.

## Spread in function calls

- It is common to pass elements of an array as arguments to a function.
- Before ES6 we used to,

```
function myFunction(a, b, c) {
  console.log(a + b + c);
}
var args = [1, 2, 3];
console.log(myFunction.apply(null, args));
```

- ES6 gives an easier way to do this using the spread operator.

```
function myFunction(a, b, c) {
  console.log(a + b + c);
}
var args = [1, 2, 3];
console.log(myFunction(...args));
```

- One more example of spread operator in function calls.

```
function myFunction(a, b, c) {
  console.log(a, b, c);
}
var args = [1, 2];
myFunction(...args, 3);
```

## Spread in Array Literals

- Before ES6, we used the following syntax to add an item at middle of an array:

```
var arr = ["One", "Two", "Seven"];
arr.splice(2, 0, "Three");
arr.splice(3, 0, "Four");
arr.splice(4, 0, "Five");
arr.splice(5, 0, "Six");
console.log(arr); // ["One", "Two", "Three", "Four", "Five", "Six",
"Seven"]
```

- You can use methords such as `push`, `splice`, `concat` to add items to an array.
- But it is easier to use the spread operator to add an item at middle of an array.

```
var arr = ["One", "Two", "Seven"];
arr.splice(2, 0, ...["Three", "Four", "Five", "Six"]);
console.log(arr); // ["One", "Two", "Three", "Four", "Five", "Six",
"Seven"]
```

## Spread in Object Literals

- In objects it copies the own enumerable properties from the provided object onto a new object.

```
const ob1 = {
  a: 1,
  b: 2,
  c: 3,
};
const ob2 = {
  d: 4,
  e: 5,
  f: 6,
};
const clone = { ...ob1, ...ob2 }; // { a: 1, b: 2, c: 3, d: 4, e: 5, f: 6
}
```

- Shallow cloning or merging objects is possible with another operator called `Object.assign()`.

## Quiz Time

- What is the output of the following code?

```
let nums = [1, 4, 1, 5];
let all = [3, ...nums, 9, 2];
console.log(all[2]);
```

- all : [3, 1, 4, 1, 5, 9, 2]

- all[2] : 4