SCHEDULED CALLBACKS

Spontaneous Right Action

Main Point Preview: Rest parameters and spread operator

Rest parameters and the spread operator are convenience syntax for, respectively, collecting function parameters into an array or assigning collection elements across a set of variables.

Function Signature (review)

- If a function is called with missing arguments(less than declared), the missing values are set to : `undefined`
- Extra arguments are ignored

```
function f(x) {
 console.log("x: " + x);
}
f(); //undefined
f(1); //1
f(2, 3); //2
```

*

No overloading!

```
function log() {
 console.log("No Arguments");
}
function log(x) {
 console.log("1 Argument: " + x);
}
function log(x, y) {
 console.log("2 Arguments: " + x + ", " + y);
}
log();
log(5);
log(6, 7);
```

- Why? JavaScript ignores extra arguments and uses undefined for missing arguments. Last declaration overwrites earlier ones.

arguments Object (legacy)

The **arguments** object is an Array-like object corresponding to the arguments passed to a function.

```
function findMax() {
 let max = -Infinity;
 for (let i = 0; i < arguments.length; i++) {
   if (arguments[i] > max) {
   max = arguments[i];
   }
 }
 return max;
}
const max1 = findMax(1, 123, 500, 115, 66, 88);
const max2 = findMax(3, 6, 8);
```

Rest parameters (ES6)

- rest parameters are only the ones that haven't been given a separate name, while the arguments object contains all arguments passed to the function
- ES6 compatible code, then rest parameters should be preferred.

```
function sum(x, y, ...more) {
 //"more" is array of all extra passed params
 let total = x + y;
 if (more.length > 0) {
   for (let i = 0; i < more.length; i++) {
   total += more[i];
   }
 }
 console.log("Total: " + total);
 return total;
}
sum(5, 5, 5);
sum(6, 6, 6, 6, 6);
```

Exercise

- write a function, multiplyEvens, that can be called with any number of arguments and returns the product of the even arguments
- do first using the arguments object
- then using …rest parameter

multiplyEvens(1, 6, 3, 4, 17, 2) → 48

Spread operator (ES6)

- The same … notation can be used to unpack iterable elements (array, string, object) rather than pack extra arguments into a function parameter.

```
let a, b, c, d, e;
a = [1,2,3];
b = "dog";
c = [42, "cat"];

// Using the concat method.
d = a.concat(b, c); // [1, 2, 3, “dog”, 42, “cat”]

// Using the spread operator.
e = [...a, b, ...c]; // [1, 2, 3, “dog”, 42, “cat”]
copyOfA = [...a]  //[1, 2, 3]

let str = "Hello";
alert( [...str] ); // H,e,l,l,o
```

Spread operator 2 (ES6)

- make a (shallow) clone of an object

```
let a, b;
a = {x:1, y:2, z:3}
b = { …a }
console.log(b) // {x:1, y:2, z:3}
b.x = 100;
console.log(a) // {x:1, y:2, z:3}
console.log(b) // {x:100, y:2, z:3} -- clone
```

Summary

*

- When we see "..."
  - can be rest parameters or spread operator
  - spread syntax "expands" an array into its elements
  - rest syntax collects multiple elements and "condenses" them into a single element
- ... In an assignment context then “rest parameters”
  - end of function definition parameters,
  - end of destructure assignment
  - gathers the rest of the list of arguments into an array.
- ... occurs in an evaluation or expression context then is “spread operator”
  - function call
  - array literal
  - expands an array into a sequence of elements
- Use patterns:
  - Rest parameters create functions that accept any number of arguments.
  - spread operator
    - spread array elements individually into another array – like concat
    - clone an array or object (shallow clone)
    - pass an array to functions that require multiple individual arguments

Main Point: Rest parameters and spread operator

Rest parameters and the spread operator are convenience syntax for, respectively, collecting function parameters into an array or assigning collection elements across a set of variables.

Main Point Preview: Timout callbacks

The asynchronous global methods setTimeout and setInterval take a function reference as an argument and then callback the function at a specified time. Science of Consciousness: Performing actions in proper sequence and correct time is critical to intelligent behavior. Actions performed from the level of pure consciousness will be spontaneously in accord with all the laws of nature.

Timers

- setTimeout allows to run a function once after the interval of time.
- setInterval allows to run a function regularly with the interval between the runs.

*

setTimeout

```
let timerId = setTimeout(func, [delay], [arg1], [arg2], ...)
```

- Func: Function or a string of code to execute.
- Delay: delay before run, in milliseconds (1000 ms = 1 second), by default 0.
- arg1, arg2… : Arguments for the function

```
function sayHi() {
  alert('Hello');
}
setTimeout(sayHi, 1000);
```

- With arguments:

```
function sayHi(phrase, who) {
  alert( phrase + ', ' + who );
}
setTimeout(sayHi, 1000, "Hello", "John"); // Hello, John
```

Pass a function, but don't run it

- Novice developers sometimes make a mistake by adding brackets ()

```
// wrong!
setTimeout(sayHi(), 1000);
```

- doesn't work,
  - setTimeout expects a reference to a function.
  - here sayHi() runs the function,
  - result of its execution is passed to setTimeout.
  - result of sayHi() is undefined (the function returns nothing), so nothing is scheduled

- function call versus function binding
  - sayHi() versus sayHi
  - execute the function versus reference to the function
  - fundamental concept!!

*

Canceling with clearTimeout

A call to setTimeout returns a “timer identifier” that we can use to cancel the execution.

```
let timerId = setTimeout(...);
clearTimeout(timerId);
```

schedule the function and then cancel it

```
let timerId = setTimeout(() => alert("never happens"), 1000);
console.log(timerId); // timer identifier
clearTimeout(timerId);
console.log(timerId); // same identifier (doesn't become null after canceling)
```

*

setInterval

The setInterval method has the same syntax as setTimeout:

```
let timerId = setInterval(func, [delay], [arg1], [arg2], ...)
```

Repeatedly calls the function after the given interval of time.

To stop further calls, we should call clearInterval(timerId).

```
// repeat with the interval of 2 seconds
let timerId = setInterval(() => alert('tick'), 2000);
// after 5 seconds stop
setTimeout(() => { clearInterval(timerId); alert('stop'); }, 5000);
```

Zero delay setTimeout

*

- There’s a special use case: setTimeout(func, 0), or just setTimeout(func).

- schedules the execution of func as soon as possible.
  - after the current code is complete.

```
setTimeout(() => alert(“Hello"), 0);
alert(“World");
```

- The first line “puts the call into event queue after 0ms”
  - scheduler will only “check the queue” after the current code is complete
  - “World" is first, and " Hello " – after it.

Exercises

- Output every second
- What will setTimeout show?

Main Point: Timeout callbacks

The asynchronous global methods setTimeout and setInterval take a function reference as an argument and then callback the function at a specified time. Science of Consciousness: Performing actions in proper sequence and correct time is critical to intelligent behavior. Actions performed from the level of pure consciousness will be spontaneously in accord with all the laws of nature.