Node JS

Chapter 2 : JSETTING SKILLS

JSETTING SKILLS PART 1

1. **Spread syntax**
2. **Destructuring Assignment**
3. **Types checking and casting**
4. **Callbacks and methods**

1. Spread syntax

Spreading arrays on another

```
// Spreading arrays on another
var list = [10, 20, 5],
  global_list1 = [5, 46, ...list],
  global_list2 = [...list, 50, 65],
  global_list3 = [5, 46, ...list, 50, 65];
console.log(global_list1, global_list2,
global_list3);
```

1. Spread syntax

Spread the arguments in function

```
// Spread the arguments in function
function max(a, b, c) {
  if (a > b && a > c) return a;
  else if (b > a && b > c) return b;
  else return c;
}
console.log(max(...list), max(...global_list1));
```

1. Spread syntax

Extending objects

```
// extending objects
var person = { firstName: "Youcef", lastName: "Madadi", age: 23 },
  P1Note = { exams: [16, 12, 14, 10] },
  person2 = { firstName: "Abdelhak", ...person, ...P1Note, age: 24 };
console.log(person2);
```

1. Spread syntax

collecting arguments as an array

```
1 //collecting arguments as an array
2 function sum(...elements) {
3   var s = 0;
4   for (var val of elements) s += val;
5   return s;
6 }
7 console.log(sum(10, 20, 30), sum(10, 30));
```

2. Destructuring Assignment

Array destructuring

```
// Array destructuring
const color = [200, 54, 96];
const [r, g, b] = color;
console.log(r, g, b);
```

Assignment separate from declaration

```
// Assignment separate from declaration
var a, b;
[a, b] = [1, 2];
console.log(a); // 1
console.log(b); // 2
```

2. Destructuring Assignment

Default values

```
// Default values
var a, b;
[a = 5, b = 7] = [1];
console.log(a); // 1
console.log(b); // 7
```

Ignoring some returned values

```
// Ignoring some returned values
const [a, , b] = [1, 2, 3];
console.log(a); // 1
console.log(b); // 3
```

2. Destructuring Assignment

Swapping variables

```
// Swapping variables
var a = 1,
  b = 3;

[a, b] = [b, a];
console.log(a); // 3
console.log(b); // 1

const arr = [1, 2, 3];
[arr[2], arr[1]] = [arr[1], arr[2]];
console.log(arr); // [1,3,2]
```

2. Destructuring Assignment

Assigning the rest of an array to a variable

```
// Assigning the rest of an array to a variable
const [a, ...b] = [1, 2, 3];
console.log(a); // 1
console.log(b); // [2, 3]
```

Object destructuring

```
// Object destructuring
const user = {
  id: 42,
  is_verified: true,
};
const { id, is_verified } = user;
console.log(id, is_verified);
```

2. Destructuring Assignment

Assigning to new variable names

```
// Assigning to new variable names
const { userId: id, is_verified: v } = { userId: 42, is_verified: true };
console.log(id, v);
```

Default value

```
// default value
const { a = 5, b = 6 } = { a: 1 };
console.log(a, b);
```

Assigning the rest of an object to a variable

```
// Assigning the rest of an object to a variable
const { age, ...names } = person;
console.log(names, age);
```

2. Destructuring Assignment

Unpacking fields from objects passed as a function parameter

```
// Unpacking fields from objects passed as a function parameter
function shoutOutFor({ first, last }) {
  console.log(
    `We would like for ${first} ${last} to come to pick up his/her package`
  );
}
var person = { first: "Youcef", last: "Madadi", age: 23 };
shoutOutFor(person);
```

3. Type checking

Type checking

```
//Type checking
var person = {
  fullName: "Youcef Madadi",
  age: 23,
  trainer: true,
  assistant: null,
};
console.log(typeof person); // object
console.log(typeof person.fullName); // string
console.log(typeof person.age); // number
console.log(typeof person.trainer); // boolean
console.log(typeof person.assistant); // object
console.log(typeof person.manager); // undefined
```

3. Type checking

Casting Types

```
// casting types
// number to strings
String(person.age);
person.age + "";
// string to number
Number("200"); // 200
Number("40d"); // NaN
// boolean to string
String(person.trainer);
person.trainer + "";
```

```
// casting types
// string to boolean
"true" == true;
"false" == false;
//number to boolean
Boolean(n); // true if n !==
0
// boolean to number
Number(false); // 0
Number(true); // 1
```

3. Type checking

Other methods for verifying

```
// checking if it is a number
Number.isNaN("20"); //false
Number.isNaN("youcef"); // true
Number.isInteger(20); // true
Number.isInteger(20.5); // false
Number.isFinite(Infinity); // false
Number.isFinite(50); // true

// checking if it is an array
Array.isArray([5, 578, 65]); // true
```

4. Callbacks and methods

callbacks

```
// callback
function Sum(arr, func) {
  var sum = 0;
  for (let i = 0; i < arr.length; i++) sum += func(arr[i], i, arr);
  return sum;
}
function fact(n) {
  return n < 2 ? 1 : fact(n - 1) * n;
}
var res = Sum([5, 4, 3, 5], fact);
console.log(res);
```

4. Callbacks and methods

Direct callback

```
// direct callback
Sum([10, 60, -50, -2], function (elm) {
  return elm < 0 ? -elm : elm;
});
```

Direct callback using Lambda functions

```
// direct callback
Sum([10, 60, -50, -2], (elm) => (elm < 0 ? -elm : elm));
```

4. Callbacks and methods

Methods that uses callbacks

```
// methods that uses callbacks
// changing array content
[10, 84, -35, 21, -12].map(function (elm, i, arr) {
  return elm * i;
});
// looping over array
[10, 84, -35, 21, -12].forEach(function (elm, i, arr) {
  console.log(elm, i, arr);
});
// sum using reduce
[10, 84, -35, 21, -12].reduce(function (acc, elm, i, arr) {
  return acc + Math.abs(elm);
}, 0);
```

4. Callbacks and methods

Methods that uses callbacks

```
// filter an array
[10, 84, -35, 21, -12].filter(function (elm, i, arr) {
  return elm < 0;
});
// confirm array using every
[10, 84, -35, 21, -12].every(function (elm, i, arr) {
  return elm > 0;
});
// confirm part of array using some
[10, 84, -35, 21, -12].some(function (elm, i, arr) {
  return elm > 0;
});
```

4. Callbacks and methods

Methods that uses callbacks

```
// Sorting an array
[10, 84, -35, 21, -12].sort(function (a, b) {
  if (a < b) return -1; // a is less than b by some ordering
criterion
  else if (a == b) return 0; // a must be equal to b
  else return 1; // a is greater than b by the ordering criterion
});
```

4. Callbacks and methods

Chain methods

```
//chain methods
[10, 84, -35, 21, -12]
  .map(function (elm, i, arr) {
    return elm * i;
  })
  .reduce(function (acc, elm, i, arr) {
    return acc + Math.abs(elm);
  }, 0);
```