Advanced Javascript

Javascript A brief background

- ECMA: European Computer Manufacturer's Association
- Compiling Javascript code

Declaring Variable

- Earlier to ES2015, we have only one option to declare a variable: var
- The const keyword:
  - Introduced in ES6
  - A const is a variable that can not be overwritten
  - Once declared, we can not change the value.
- The let keyword
  - Javascript now has lexical variable type.
  - The variable declared with let keyword, we can scope a variable to the code block.
  - Protects the value of global variables.

Template Strings

- Alternative to string concatenation
- Allows us to insert values in between the string
- We can create one string and insert the variables using “${ }”.
- Template string honours whitespaces.

Creating Functions

- In case of repetitive task, we can go for functions
- Function declaration:
  - Starts with the keyword **function**
  - Followed by name of the function
  - Parameter list
  - Block of code to be executed

Function Expressions

- This is another option to define a function
- Creating a function as Variable
- Ex:

```
const logMethod=function(){
        //Code to execute
                }
```

Invoking function: logeMethod();

Function Expressions – A Note

- Function declarations are hoisted.
  - We can invoke the function before declaring the function in code.
  - We can not invoke the function before its declaration using Function Expression

Passing Arguments & Returns

- We can pass named parameters to the function using parenthesis in function declaration
- The return statement specifies the value to be returned.
- Default parameters:
  - Similar to C#, we can provide default values for the function parameters.

Default Parameters

- ES6 supports default parameteers

```
Const defPerson={
name:{fname:'sai', lname='durga'},
skill:'C#'
}

function logActivity(person:defPerosn)
{
Console.log(person.name.fname);
}
```

Arrow Functions

- With arrow function we can create functions with out the keyword ‘function’.
- We can skip the return keyword also.

Traditional Way:

```
 function Greetings(name)
{
 return ‘Hi’+name;
}
```

Can be replaced with:

```
 const grretings = (fname)=> ‘Hi’+fname;
```

Arrow functions – Returning Object

```
 const createPerson=(fname,lname)=>({firname:fname,lastname: lname});
Console.log(createPerosn(‘sai’,’durga’);
```

Destructuring Objects

- Destructing assignment of an object allows us to locally scope fields of an object

-
  ```
  const sandwich={
              bread: 'Italien Bread',
              cheese: 'swiss',
              toppings: ['lettuce','Jalpinoes','tomato']
          }
  ```

```
const {bread,cheese}=sandwich;
console.log(bread,cheese);

bread='sweedish bread';
```

The code pulls bread and cheese properties out of object and assign to local variables.

We can change the value of the local variable WITHOUT effecting the object.

Destructuring incoming input arguments

```
const regularPerson={fname:'sai', lname:'durga'}


const myFunction= regularPerson=>{ console.long(regularPerson.fname)};
        myFunction(regularPerson);
```

Digging deep into Object

```
const myFunction2=({fname})=>{console.log(fname)};
        myFunction2(regularPerson);
```

Argument destructing

Destructuring Arrays

- Values can be destructuring from arrays.

```
const [firstAnimal]=[ ‘Horse’, ‘’Mouse’, ‘Cat’]; //’Horse’
const[ ,  , thirrdAnimal]=[ ‘Horse’, ‘’Mouse’, ‘Cat’]; //Cat
```

Object Literal Enhancements

- Object literal enhancement is the opposite to the destructering
- Process of re-structuring / putting back the object.
- We can grab the variables from global scope and add them to object.

```
const name='Tallac';
const elevation=9738;

const funHike={name,elevation}
```

name and **elevation** are the keys of the **newly created** object

Console.log(funHike.name);

The SPREAD operator

- The spread operator **(…)** the three dot syntax performs several tasks.
- Used to combine content of arrays to create third array.

```
const numSeries1=[1,3,5,7]
const numSeries2 = [2,4,6,8]
const combinedSeries=[…numSeries1,…numSeries2];
```

The SPREAD operator – Application 2

- We can reverse the array without changing / effecting the ordinal of original array

```
const numSeries1= [1,2,3,4,5,6]
const numSeries2=[…numSeries1].reverse();
```

Here we used SPREAD operator to copy the array

The SPREAD operator – Application 3

- Used to pick the remaining items of an array.

```
const numSeries1= [1,2 3, 4, 5, 6]


 const [first, …others]=numSeries1
 console.log(first);                  // prints : 1
 console.log(others);                 // Prints : [2, 3,4,5,6]
```

The SPREAD operator – Application 4

- Used to accept the function parameter as an array (Variable number of arguments)

```
 function directions(...args)
{
  let [irst, ...others]=args;
 console.log(first);
 console.log(others);
}
```

Invoking the function:

```
 directions(1,2,3);
 directions(1,2,3,4);
```

The SPREAD operator – Application 5

- The SPREAD operator can be used with objects also

const morning={ breakfast: ‘oatmeal’, lunch:’ South Indian Meal’}

const dinner= “North Indian Thali”;

const **mealPack**= {…morning, dinner};

console.log(**mealPack**);

The out put will be:

```
{
breakfast:”oatmeal”,
lunch:”South Indian Meal”,
dinner:”North Indain Thali”
}
```

Q & A