More About Functions in JavaScript

Monday, 28 August 2023 11:52 PM

→ IIFE : This is immediately invoked function expression. It is also known as self-executing fn.

```
4  (function square(num) {
5    console.log(num * num);
6  })(5);
7
```

→ So this fn is invoked as soon as it is encountered

eg:

```
4  (function (x) {
5    return (function (y) {
6      console.log(x);
7    })(2);
8  })(1);
9
```

→ output is: 1

→ Now, this is an IIFE, so it gets invoked as soon as its encounters.

→ since, the inner fn creates a closure with its parent, thus, 'x' is printed as 1, because inner fn has access to the lexical environment of its parent.

Q.

```
4  var x = 21;
5
6  var fun = function () {
7    console.log(x);
8    var x = 20;
9  };
10
11 fun();
12
```

→ o/p : undefined.

→ This is because an execution context is created whenever a fn is created.

→ var is fn scoped so, its value is undefined.

→ Difference between arrow fns & normal fns?

1)

```
4  // 1 - Syntax
5  function square(num) {
6    return num * num;
7  }
8
9  const square = (num) => {
10   return num * num;
11 };
12
```

→ Syntax.

2) Arrow fns have an implicit "return" keyword.

```
13 // 2 - Implicit "return" keyword
14 const square = (num) => num * num;
```

3) Arrow fns do not have the 'arguments' object unlike normal fns. 'arguments' is an array-like object which is not exactly an array, i.e it behaves like array but we can't use array methods with it.

```
16 // 3 - arguments
17 function fn() {
18   console.log(arguments);
19 }
20
21 fn(1, 3, 2);
```

```
script.js:18
Arguments(3) [1, 3, 2, callee: ƒ, Symbol(Symbol.iterator): ƒ] ℹ
  0: 1
  1: 3
  2: 2
▶ callee: ƒ fn()
  length: 3
▶ Symbol(Symbol.iterator): ƒ values(
▶ [[Prototype]]: Object
```

(output in case of normal f^n's)

So, we got all the arguments, without even receiving them as parameters.

```
23  const fnArr = () => {
24    console.log(arguments);
25  };
26
27  fnArr(1, 3, 2);
28
```

```
▼Uncaught                          script.js:24
ReferenceError: arguments is not
defined
    at fnArr (script.js:24:15)
    at script.js:27:1
 fnArr        @ script.js:24
 (anonymous)  @ script.js:27
```

(o/p in case of arrow f^n's)

4) 'this' keyword behaves differently in case of both types of f^n's.

eg:

```
29  // 4 - "this" keyword
30  let user = {
31    username: "Roadside Coder",
32    rc1: () => {
33      console.log("Subscribe to " + this.username);
34    },
35    rc2() {
36      console.log("Subscribe to " + this.username);
37    },
38  };
39
40  user.rc1();
41  user.rc2();
42
```

→ So, first gives o/p as:
Subscribe to undefined
and second gives o/p as:
Subscribe to Roadside Coder

This is because in case of a normal f^n, 'this' keyword points to its local object while in case of an arrow f^n, 'this' keyword points to the global object.