Apptron Technologies

ES6 Functions

Functions are the building blocks of readable, maintainable, and reusable code. Functions are defined using the function keyword.

```
function function_name() {
  // function body
}

function_name()
```

```
//define a  function
function test() {
  console.log("function called")
}
//call the function
test()
```

Classification of Functions

Functions may be classified as Returning and Parameterized functions

1) Returning functions

Functions may also return the value along with control, back to the caller. Such functions are called as returning functions.

```
function function_name() {
  //statements
  return value;
}
```

A returning function must end with a return statement.

A function can return at the most one value. In other words, there can be only one return statement per function.

The return statement should be the last statement in the function.

```
function retStr() {
  return "hello world!!!"
}
var val = retStr()
console.log(val)
```

2) Parameterized functions

Parameters are a mechanism to pass values to functions. Parameters form a part of the function's signature. The parameter values are passed to the function during its invocation.

```
function func_name( param1,param2 ,.....paramN) {
  ......
  ......
}
function add( n1,n2) {
  var sum = n1 + n2
  console.log("The sum of the values entered "+sum)
}
add(12,13)

function add(a, b = 1) {
  return a+b;
}
console.log(add(4)
```

Anonymous Function

Functions that are not bound to an identifier (function name) are called as anonymous functions. These functions are dynamically declared at runtime. Anonymous functions can accept inputs and return outputs, just as standard functions do. An anonymous function is usually not accessible after its initial creation.

a) Anonymous functions

```
var res = function( [arguments] ) { ... }
```

```
var f = function(){ return "hello"}

console.log(f())
```

a) Anonymous Parameterized Function

```
var func = function(x,y){ return x*y };

function product() {
  var result;
  result = func(10,20);
  console.log("The product : "+result)
}

product()
```

The Function Constructor:

The function statement is not the only way to define a new function; you can define your function dynamically using Function() constructor along with the new operator.

```
var variablename = new Function(Arg1, Arg2..., "Function Body");
```

a) Function Constructor

```
var func = new Function("x", "y", "return x*y;");
function product() {
  var result;
  result = func(10,20);
  console.log("The product : "+result)
}
product()
```

result :

```
The product : 200
```

Lambda Functions

Lambda refers to anonymous functions in programming. Lambda functions are a concise mechanism to represent anonymous functions. These functions are also called as Arrow functions.

a) Parameters : A function may optionally have parameters.

2) The fat arrow notation/lambda notation (=>): It is also called as the goes to operator.

3) Statements : Represents the function's instruction set.

```
([param1, parma2,…param n] )=>statement;
```

```
var foo = (x)=>10+x
console.log(foo(10))
```

result :

20

a) Lambda Statement

```
var msg = ()=> {
  console.log("function invoked")
}
msg()
```

result :

function invoked

b) Syntactic Variations

Optional parentheses for a single parameter.

```
var msg = x=> {
```

```
  console.log(x)
}
msg(10)
```

```
c)
function add2(num) {
  return num + 2;
 }
add2(2);
// 4
```

```
d)
const add2 = num => num + 2
```

Arrow Functions have two main benefits:

a) A shorter syntax than typical functions

b) No binding of this

Function Expression Syntax

A function expression is an anonymous function object that we set equal to a variable. It is anonymous because the function has no name. The syntax consists of the function keyword, followed by zero or more parameters, and concludes with the function statements

```
const name = function(parameters){
  statements
```

```
}

const double = function(num){
  return num * 2;
}

double(3);
```

Arrow Function Syntax

An arrow function looks similar to a function expression it's just shorter. Again we assign an anonymous function to a named variable. The syntax of the arrow function consists of zero or more parameters, an arrow => and then concludes with the function statements.

```
const name = (parameters) => {
  statements
}

const double = (num) => {
  return num * 2;
}

const doubleEXP = function(num){
            return num * 2;
}
// Arrow Function
```

```
const doubleARR = (num) => {
        return num * 2;
}

doubleEXP(3);
// 6
doubleARR(3);
```

We're making progress, but our arrow function can actually be simplified much more. So much more in fact, that the double function written above can also be written like this

```
const double = num => num * 2;
```

```
const double = (num) => {
  return num * 2;
}
// Is the same as
const double = num => num * 2;
```

Arrow Function Parameters:

With arrow function parameters, it's important to remember that: The number of parameters in an arrow function affects the syntax of that function

If there are 0 Parameters, arrow functions use empty parenthesis:

```
() => { statements }
```

```
parameter => { statements }
```

If there are 2+ Parameters, parameters go inside parenthesis:

```
(param1, param2, ...) => { statements }
```

```
const double = (num) => {
  return num * 2;
}
// Is the same as
const double = num => num * 2;
```

Arrow Function Return:

Arrow functions also have a built in way to shorten a return syntax: If an arrow function is simply returning a single line of code, you can omit the statement brackets and the return keyword. This tells the arrow function to return the statement.

```
let name = function(parameters){
  return expression
}
```

```
let name = (parameters) => expression
```

```
const double = (num) => {
  return num * 2;
}
// Is the same as
const double = num => num * 2;
```

a)

```
const add3 = function(num1, num2, num3){
  return num1 + num2 + num3;
}
```

b)

```
const square = function(num){
  return num ** 2;
}
```

c)

```
const sayHi = function(){
  console.log('Hi');
}
```

```
const add3 = (num1, num2, num3) => num1 + num2 + num3;
```

```
const sayHi = () => {
  console.log('Hi');
}
```