Fun functions –6 (51 coding exercises)

“Example is the best precept.”— Aesop

6.1 Functions as objects

Exercises

1. What type of object is a function?

2. What are some properties of the in-built function object?

3. Answer the following about function expression employees:

    - Access the `name` property of `employees`. What is the value of the `name` property of employees?
    - Access the `length` property of employees. What is the value of the `length` property of employees?
    - Add a `num` property to the function whose value is equal the parameter `x`
    - Add a method to the function expression called `salary`. In the company ‘Dev shop foo’ everyone is paid $89,000 per year regardless of their position in the company. Calculate the total cost when there are 79 employees

```
let employees = function information(x) {
 let company = 'Dev shop foo';
 employees.num = x;
 if (x > 100) {
  console.log(`${company} is a mid sized company`);
 } else if (x < 100) {
  console.log(`${company} is a small sized company`);
 } else {
  console.log(`${company} is a large sized company`);
 }
}
```

4. Why are functions called objects?

Answers

1. A function is a special kind of object called a function object
2. length, name, caller
3. function expression `employees`:

   The value of the `name` property is:

   ```
   console.log(employees.name); // "information"
   ```

   The value of the `length` property is:

   ```
   console.log(employees.length); 1
   ```

   `salary()` method which calculates the total salary of all employees

   ```
   employees.salary = function(){
     let salary = employees.num * 89000;
     console.log(salary)
   }
   employees(79);
   employees.salary(); //7031000
   ```

4. This is because you can work with function objects as though they were objects. Functions may be assigned to objects, passed in as arguments to other functions, returned from other functions

6.2 Functions

Exercises

1. Declare a function called `oddNum`. The function takes 1 parameter called `x`. The function does nothing as yet.

2. In the body of the function `oddNum()`, do the following:

   - The purpose of the function is to determine whether a passed in argument x is an odd number.
   - If a passed in argument is an odd number console.log the statement (`x + 'is an odd number'`
   - If a passed in argument is not an odd number console.log the statement (`x + 'is not an odd number'`

3. Declare a function called `prime`. The function takes 1 parameter called x. The function is empty and does nothing as yet.
4. In the body of the `prime()` function, do the following:

   - Evaluate whether the passed in argument is a prime number or not.
   - The function will console.log the string x + 'is a prime number' if a passed in argument is a prime number
   - If the argument is not a prime number, it will console.log the string x + 'is not a prime number'.
   - **Note:** A prime number is a natural number greater than 1 that has no divisors other than 1 and itself.

5. Determine the value of **this** in the following function:

```
function outer(){
  'use strict';
  let x = 10;
  function inner(x){
    let y = x + 10;

    console.log(this);

  }

  inner();

}

outer();
```

6. Code a function which will calculate the factorial of any number passed to it as an argument

7. Write a function which accepts an array as a parameter. The function should do two things:

    - Check if the parameter passed to it strictly an Array
    - Remove duplicates from the array if any

8. Write a function to swap the values of two variables a and b

    - It must have 3 property: values pairs (`name, genre, year`) and one method called `introduction` which prints out the name of the author and the book genre. You can do this with and without the **this** keyword
    - Call the introduction method on the author object using the **dot notation**

9. What is the difference between a function parameter and argument?

10. Analyze the following block of code, what do you think will be logged to the console?:

```
function outer() {

      let inner = function(x) {

            console.log(x)

      }

      inner();

}

outer();
```

11. How do you think we should rectify the code in question #10 (above) such that the `console.log` statement should log the value of `x`
12. Code a function called `stringVal()` with the following requirements:

    - The function takes 3 parameters `a`, `b` and `c`
    - The function will log to the console the values of `a`, `b` and `c`
    - The function should take in the following string arguments: `('Hello', 'there', 'world', '!')`
    - What do you think is logged to the console and why?

13. For the preceding function, write a statement of code that will `console.log` the number of arguments passed to a function:

```
function clients([a, b, c], [x, y]) {
  // write your code here
}
clients([1,2,3],['a','b']);
```

14. Have a look at the following function `someMath()` and try and gauge what is returned:

```
function someMath(a,b,c){
  return(a * b + c);
}
someMath(10,13);
```

Answers

1. function `oddNum(x)`

```
function oddNum(x) {


}
```

2. function `oddNum(x)`

```
function oddNum(x) {
     if(x % 2 === 0) {
            console.log(x + ' is not an odd number');
      } else {
            console.log(x + ' is an odd number');
      }
}
oddNum(3);
"3 is an odd number"
```

3. function `prime(x)`

```
function prime(x) {


}
```

4. function `prime(x)`

```
function findPrime(x) {
      let num = 0;
      //check if the x is divisible by itself and 1
      for(i = 1; i <= x; i++) {
            if(x % i == 0) {
                  //increment value of num
                  num++;
            }
      }
      if(num == 2) {
            console.log(x + ' is a Prime number');
      } else {
            console.log(x + ' is NOT a Prime number');
      }
}
```

```
findPrime(5);
```

5. In strict mode, the value of **this** is `undefined`
6. function `factorial(num)`

```
function factorial(num) {
      let result = num;
      if(num < 0) {
            return false;
      } else if(num === 1 || num === 0) {
            return 1;
      } else {
            while(num >= 2) {
                  result = result * (num - 1);
                  num--;
            }
            return(result);
      }
}
factorial(5);
```

7. function `checkArray(x)`

```
function checkArray(x){
  if(Array.isArray([x])){
        return x.filter((a, b) => x.indexOf(a) === b)
  }
}
  console.log(checkArray([1,1,1,20,3,3])); // [1, 20, 3]
```

The `indexof()` method finds the first index of an item in an array. The `filter()` method will return an array of all the elements that pass a test.

8. function `swap (num1, num2)`

```
function swap(num1, num2){
 let temp = num1;
 num1 = num2;
 num2 = temp;
 console.log(num1, num2);
}
swap(10, 30);// 30, 10
```

9. A parameter is passed into the function declaration. It is a placeholder for a real value to be passed into the function later on when the function is called. A function argument is value that is passed into the function call.
10. Arguments that are not passed in will have a value of undefined
11. We can pass in an argument to the function call of inner():

```
function outer(){
let inner = function(x){
    console.log(x)
  }
inner(10);
}
outer();
```

This will log to the console the numeric value `10`:

```
10
```

12.

```
function stringVal(a,b,c) {
  console.log(`${a} ${b} ${c}`);
}
stringVal('Hello', 'there', 'world', '!');
```

The following is logged to the console:

```
"Hello there world"
```

When the number of arguments are not the same as the number of parameters defined in a function, the extra arguments will be ignored

13.

```
function clients([a, b, c], [x, y]) {

  console.log(arguments.length);

}

clients([1,2,3],['a','b']);
```

The following will be logged to the console, as there are two array parameters passed to the function:

```
2
```

14. The answer is `NaN`. if we pass in one argument less than the number of parameters The parameters that have no corresponding arguments are set to `undefined`. The mathematical operation `10 * 13 + undefined` is equal to `NaN`.

6.3 Function expressions

Exercises

1. A function expression cannot be used unless it is defined. True/False
2. Function expressions are hoisted. True/False
3. Write a function expression such that:

    - The function is referenced by a variable declared as `multiply`
    - The function takes 3 parameters `x`, `y` and `z`
    - The function will return the value of `x`, `y` and `z` multiplied with each other
    - Call the function expression with the following arguments: `2, 20, 10`

4. Have a look at the following code block and then answer the questions:

```
let multiply = function(x, y, z) {

  return(x * y * z);

}

multiply(2,20,10);
```

```
function collection() {
  let fruit ={
    a: 'apple',
    b: 'banana',
    c: 'cantaloupe',
  }
  console.log(fruit);
}
```

Calling the function `collection()` before it’s declaration. What is logged and why?

```
collection();
let multiply = function(x, y, z) {
  return(x * y * z);
}
multiply(2,20,10);

function collection() {
  let fruit ={
    a: 'apple',
    b: 'banana',
    c: 'cantaloupe',
  }
  console.log(fruit);
}
```

Calling the function `multiply()` before it’s declaration. What is logged and why?

```
multiply ();
let multiply = function(x, y, z) {
  return(x * y * z);
}
multiply(2,20,10);

function collection() {
  let fruit ={
    a: 'apple',
    b: 'banana',
    c: 'cantaloupe',
  }
  console.log(fruit);
}
```

5. In this exercise we will make our own timer using the `setInterval()` method and Date constructor:

    - Declare a function expression called `timer`
    - The function should log the current local time to the console.
    - ***Hint:*** use the `new Date()` constructor and from the date, then get thetime with the `toLocaleTimeString()` method
    - Pass in the `timer` function expression to the `setInterval()` method as an argument
    - This should log to the console the time at intervals of 1 second

Answers

1. True.
2. False
3.

```
let multiply = function(x, y, z) {
  return(x * y * z);
}
```

```
multiply(2,20,10); //400
```

4. Calling the function `collection()` before its declaration will log to the console the following, as function declarations are hoisted:

```
Object {
  a: "apple",
  b: "banana",
  c: "cantaloupe"
}
```

Calling the function `multiply()` before its declaration will log the following `ReferenceError` to the console, as function expressions are not hoisted:

```
ReferenceError: can't access lexical declaration
'multiply' before initialization
```

5.

```
let timer = function(){
let date = new Date();
let timeNow = date.toLocaleTimeString();
console.log(timeNow)
}
setInterval(timer, 1000);
```

6.4 Immediately invoked function expressions (IIFE)

Exercises

1. What is an IIFE and what is a use case for using IIFEs?
2. Code an IIFE that will console.log the string `'hello world'`
3. Why are IIFE's used?
4. What is the context of **this** in the following function:

```
(function (){
     console.log(this);
})();
```

5. Have a look at the following code and do the following tasks:

    - Convert the function `logPerson` to an IIFE
    - What will be logged to the console first?

```
let person = 'Rena'
function logPerson(){
  let person = 'Robbie'
  console.log(person);
};
logPerson();
console.log(person);
```

The following is logged to the console currently:

```
"Rena"
"Robbie"
```

6. Have a look at the following block of code. What will be logged to the console inside the IIFE and outside the IIFE?

```
let x = 10;
(function(x){
  x++;
  console.log(x);
})(x);
console.log(x);
```

Answers

1. An IIFE is an immediately invoked function expression. It is executed immediately. Use cases of IIFEs are:

- Loading a user’s preferences, shopping cart when they return to website/web application by using an IIFE
- 2. Showing the user a message as soon as they visit a website

2.

```
(function(){
  console.log('hello world');
})();
```

3. To avoid polluting the global namespace, as all the variables used inside the IIF are not visible outside its scope.
4. The context of **this** will be the global window object.
5.

```
let person = 'Rena';
(function(){
  let person = 'Robbie'
  console.log(person);
})();
console.log(person);
```

The following will be logged to the console in this order:

```
"Robbie"
"Rena"
```

6. The number `11` is logged to the console inside the IIFE and the number `10` is logged to the console outside of the IIFE. Any variables declared or changed inside an IIFE cannot be accessed outside. Hence, there are two different values logged to the console for `x`.

```
11
10
```

6.5 Anonymous functions

Exercises

1. What is an anonymous function and can you think of any popular/common examples where anonymous functions are used in JavaScript?
2. Declare a function expression referenced by a variable called `fooScore` and do the following:
    - The function takes one parameter `x`
    - Inside the function raise `x` to the power of 5
    - Invoke the function and pass in the number 2 as an argument
    - Hint: use `Math.pow()` to raise `x` to the power of 5
3. Let's code our own count down timer. Use the `setInterval()` method to countdown from 10 to 0 at intervals of 2 seconds.
4. Use the `window.onload()` method to execute a function which will alert the string `'Hello User'` when a page loads

Answers

1. A function that does not have a name is an anonymous function. IIFes, setTimeout() functions, and a lot of DOM event handlers use anonymous callback functions.
2.

```
let fooScore = function(x){
  return(Math.pow(x, 5));
};
fooScore(2); //32
```

3.

```
let count = 10
setInterval(function(){
  if(count >= 0){
```

```
  console.log(count);
  }
 count--

}, 2000)
```

4.

```
window.onload = function(){
  alert('Hello User!')
}
```

6.6 Arrow functions

Exercises

1. Convert the following function to an arrow function

```
function employee(hours) {
return hours;
}
employee(10);
```

2. Convert the following function to an arrow function:

```
function employee(hours) {
hours *= 1.5;
return hours;
}
```

3. When should you use arrow functions?

4. Fix this function so that the the `setInterval()` method logs numbers from 0 onwards:

```
function timer() {
  seconds =0;
  setInterval(()=>{
    console.log(this.seconds++);
  }, 1000)
}
timer(); NaN
```

5. Analyze the following code, what do you think will be returned when the method `productDetails()` is called?

```
let headset = {
    model: 'Oculus Quest',
    company: 'Facebook',
    productDetails: () => {
    return `${this.company}${this.model}`;
    }
};

console.log(headset.productDetails());
```

Answers

1.

```
let employee = (hours) => hours;
employee(10); // 10
```

2.

```
let employee = (hours) => {
hours*= 1.5;
return hours;
}
employee(190);
```

3. Arrow functions should be used when you're aiming for shorter concise syntax. Arrow functions use , **this** from their own enclosing scope. Therefore, **this** context within the arrow functions is lexically or statically scoped .This provides a narrower and safer scope.
4.

```
function timer() {
  seconds = 0;
  setInterval(()=>{
    console.log(this.seconds++);
  }, 1000)
}
timer();
```

Remember that an arrow function does not have its own binding of **this**. Instead **this** is lexically bound. Which means arrow functions use **this** from their own enclosing scope.

5. `undefined`

6.7 Rest syntax

Exercises

1. What is the purpose of the rest parameter?

2. Have a look at the following function with 9 parameters. Re-factor the function using rest syntax:

```
function adder(a,b,c,d,e,f,g,h,i){
  console.log(arguments.length)
  for(let i = 0; i < arguments.length; i++) {
    console.log(arguments[i] + arguments[i])
  }
}
```

```
adder(1, 2, 3, 4, 5, 6, 7, 8,9);
```

The following is logged to the console, thereby indicating that each argument is doubled.

```
2

4

6

8

10

12

14

16

18
```

Note: *Remember that you can use array methods as the parameters are converted into an array when using the rest syntax*

3. Have a look at the following function with 9 parameters. Re-factor the function using rest syntax. The function sums up all arguments passed to it:

```
function sum(a, b, c, d, e, f, g, h, i){

 console.log(a + b + c + d + e + f + g + h + i);

}

sum(1, 2, 3, 4, 5, 6, 7, 8, 9);
```

The function will log the following sum:

```
45
```

4. Analyze the following function what is returned? If an error is thrown, how will you fix it?

```
function workshopAttendees(a,...b, c){

 return true;

}
```

5. Can you think of any differences between a rest parameter and the arguments object which is inherently present in all functions?

6. What is the different between rest and spread parameters?

7. Write a function that will multiply any amount of arguments passed to it by 8

Answers

1. The rest parameter introduced in ES6 is used to pass in an indefinite number of parameters to a function. These parameters are represented as an array; therefore you can use array methods on them.

2.

```
function adder(...a) {

  return a.forEach(function(item){

    console.log(item + item);

  })

}


adder(1, 2, 3 , 4, 5, 6, 7, 8, 9);
```

3.

```
function sum(...params) {

   return params.reduce((previousIndex, currentIndex) =>
{

    return previousIndex + currentIndex;

  });

}

console.log(sum(1, 2, 3, 4, 5, 6, 7, 8, 9)); // 45
```

4. An error is thrown as only last parameter is allowed to be defined with the rest syntax:

```
SyntaxError: parameter after rest parameter
```

In order to fix it, change the rest parameter to be the last aparameter passed to the function:

```
function workshopAttendees(a, c, ...b){
return true;
}
//we can then call the function:
workshopAttendees(1,2,3,4,5,6,7);
```

5. There are a couple of differences between the two:

    - The arguments object is array like but not an array. Whereas the rest parameters are array instances which is why you can use array methods such as sort(), map(), filter() etc. on them
    - The arguments object contains all the arguments passed to an object. Whereas rest parameters are a reference to arguments to be passed later on in the function and are not formally defined in a function

6. Example of the rest parameter:

```
function rest(x,y,...z){
    console.log(x);  // 1
    console.log(y);  // 2
    console.log(z);  // [3,4,5]
}


rest(1,2,3,4,5);
```

Example of the spread operator:

```
let array1 = ['a', 'b', 'c'];
let array2 = ['d', 'e', 'f'];
let combinedArray = [...array1, ...array2];
console.log(combinedArray);//["a","b","c","d","e","f"]
```

The most obvious different is that the rest operator is used with functions , and the spread operator is used with arrays. The rest operator collects all the remaining elements into an array. Whereas, the spread operator allows arrays/objects/strings to be expanded into single arguments/elements.

7.

```
function multiplier(...a){
  a.forEach(function( item){
   return(item * 8);
 });
}
multiplier(1, 8, 2, 10, 9); // 8,  64, 16, 80, 72
```

6.8 Callback functions

Exercises

1. What is a callback function?
2. Finish the following code snippet. The declarations for onSuccess and onFailure callback functions are missing. You may add anything inside the body of the callback functions

```
function respond(onSuccess, onFailure) {
    let error = true;
    if (error) {
      onFailure(error)
    } else {
      onSuccess()
    }
  }
```

```
respond(onSuccess, onFailure);
```

3. Have a look at the following code and explain what the order of the console.log statements will be:

```
console.log(1);

console.log(2);

setTimeout(() => {

  console.log(3);

}, 0);

console.log(4);
```

4. Can you think of a callback function used in the DOM?
5. Callback functions can be confusing, nested callbacks are often called callback hell. Can you explain what is happening here?

```
function a(x){

  b(c);

}

function b(y){

  c();

}

function c(){

  console.log('hi')

}

a(b);
```

6. Code a function called `input` which takes a parameter called `text` and a callback function as a second parameter. When the function `input()` is called it should return a string argument passed to it reversed. For example the string `'hello world!'` is returned as `'!dlrow olleh'`

Answers

1. A function that is passed as an argument to another function is called a **callback** function. The function that accepts another function as an argument is called a **higher-order** function and contains the logic for when the callback function will be executed. This outer function undertakes the asynchronous task
2.

```
function respond(onSuccess, onFailure) {
    let error = true;
    if (error) {
      onFailure(error)
    } else {
      onSuccess()
    }
  }
function onFailure(err){
  console.log(err)
}
function onSuccess(err){
  console.log('Success!')
}
respond(onSuccess, onFailure);
```

3. The following is the order of the statements logged to the console:

```
1
2
4
3
```

Notice that the `setTimeout()` function has a `0` second delay. However, the `setTimeout()` function does not execute immediately. This is because callback functions are treated as second class citizens. They are executed after regular functions (first class citizens) finish executing first.

4. The `addEventListener()` method is an example of a callback function that is called in response to a click event:

```
document.getElementById('button').addEventListener('click',
function(){

 console.log('Button clicked');

});
```

5. Function `a()` takes a parameter `x` and function `b()` is called from inside this function. Similarly function `b()` takes a parameter called `y` and function `c()` is called from inside fuction `b()`. Lastly, function `a(b)` is called passing in function `b` as an argument.

6.

```
function input(text, reverse){

  reverse(text);

}

function reverse(string) {

  let reverse = string.split('').reverse().join("");

  //console.log(reverse);

  return reverse;

}

input('hello world!', reverse);
```