100 Coding exercises

" You cannot dream yourself into a character; you must hammer and forge yourself one." —Thoreau

Exercises

1. Write a recursive function to countdown from 0 - 10
2. Will be the output of the following:

```
for (var i = 0; i < 5; i++) {
}
console.log(i); ??
```

3. How do you change the following code so that `obj1.print()` outputs `6` instead of `7`

```
var obj = {
    a : 6,
    print: function () {
        console.log(this.a);
    }
};
var obj1 = {
    a : 7,
    print: null
};
obj.print(); //->6
```

4. Code a function to find if a given year passed to it as an argument is a leap year
5. What block scope keyword when changed produces the above console.log output?

```
const myString = 'Hello 2020';

 myString = 'Hello World!';

 console.log(myString)//"Hello World!"
```

6. What will be logged to the console?

```
let games = ["King's Quest", "Super Mario", "Zelda",
"Contra"];

games.splice(2, 0, "Doom", "World of Warcraft");

games.slice(0,2);

console.log(games); // ? What will this be
```

7. `innerHTML` can increase the possibility of a XSS attack. What is an alternative for setting text?

8. Consider the following code. What will be displayed on the webpage (inside the `<p>`)

```
  <p id="myP"></p>
```

```
let num = 10;

 function sum( ) {

  let  num = 100;

  document.getElementById('myP').innerHTML =(num+num);
// 20 or 200?

   }

sum();
```

9. Analyze the following code, what will be logged to the console?

```
let langs =["JavaScript", "C#", "Rust", "C++", "Python"];

  delete langs[3];

  console.log(langs.length);

  console.log(langs[3]);
```

10. Empty the following array:

```
let arr1= ["", "A", 9, "C++", false];
```

11. What will the contents of the array `fooBar` be?

```
let fooBar = new Array(5);

fooBar[2] = "Hello";

console.log(fooBar); // ?
```

12. Re-write this function in run in the browser, with the same functionality:

```
(function IIFE(){

    alert( 'hello world!' );

})();
```

13. White space is retained in template literals. True or False?

14. Remove the first element in the following array. What principle is used?

```
let myQueue = [];

myQueue.push('a');

myQueue.push('b');

myQueue.push('c');
```

15. Create a shallow copy of the following object?

```
let keys = {

  a: 1,

  b: 2,

  c: 3,

  d: 4

};
```

16. Carrying on from the above question, what happens to the shallow copy when `keys.b = 20` ?

17. What is the difference between a shallow and deep copy?

18. Code a function so that the `even` array has only even numbers from the `nums` array and the `odd` array only contains odd numbers from the `nums` array:

```
let nums = [1,2,34,54,55,34,32,11,19,17,54,66,13];
```

```
let even = [];
var odd = [];
```

19. Print/console.log numbers up to 100 that are divisible by both 3 and 5
20. True or false:

```
let c = 0;
let d = false;
console.log(c !== d);
```

21. Create a function which will accept two octal numbers as arguments and return their sum. The function should not generate errors while in strict mode
22. Create a function that returns an error while in strict mode
23. Explain the output of the following:

```
(function() {
   b = 2;
   var a = b;
})();
```

24. For the following estimate what the console.log statements will be:

```
let name = 'Jan Pan';
let person = {
   name: 'Kira Doe',
   details: {
      name: 'Penelope Bruze',
      print: function() {
         return this.name;
      }
   }
};
console.log(person.details.print());
```

```
let detailsPrint = person.details.print;
console.log(detailsPrint());
```

25. Find the common elements from the following 3 arrays:

```
let num1 = [1, 2, 3, 4];
let num2 = [10, 20, 30, 1];
let num3 = [100, 200, 300, 1];
```

26. What is logged to the console? True or false:

```
let students = 0;
 let classroom = false;
 if(students > 10) {
    let classroom = true;
   }
console.log(classroom);
```

27. What will be logged to the console?

```
let mySymbol = Symbol('kode');
let mySymbol2 = Symbol('kode');
console.log(mySymbol === mySymbol2)// true or false.
```

28. Use `forEach()` to multiply every item in the `nums` array by 2 and push it to the `newNums` array. Also log its index at each iteration:

```
let nums = [1,2,3];
let newNums =[];
```

29. What does the word **this** mean within a function? Explain with an example.

30. Arrow function do not have a **this** context. True or False?

31. Analyze the following code and determine what `console.log` will return

```
let x = () =>{};
```

```
let y= {};

console.log(typeof(x) !== typeof(y)); //true or false
```

32. What will the output be?

```
let a = 30;

let b = a++;

let c = ++a;

console.log(a,b,c);
```

33. Code a function to generate a random number between 1-100

34. Code a function to return a random array item when the function is called:

```
let items = [1, 10, 79, 808, 33, 45, 19800];
```

35. Code a function to find if a given number is to the power of 2

36. What 3 ways can you think of to empty the following array:

```
let alphas =['a', 'b', 'c', 'e', 'k'];
```

37. Examine the following code and estimate the length and contents of `array2`:

```
let array1= ['a','b','c','d','e','f']; // Created array

let array2 = array1;

array1.push('k');
```

38. Code a function to mask an email address after 5 characters. For example: abcde12345@gmail.com is abcde…..@gmail.com

39. Code a function called `course` which will be a constructor function for objects. Pass the following parameters: `duration`, `students`, `instructor`. Create 3 new objects called `IntroductionToHTML`, `IntroductionToCSS` and `IntroductionToSass` using the course function constructor. You may pass in any values that you like to the objects as values for the duration, students and instructor arguments.

40. Carrying on from #39, delete the duration property for the `IntroductionToHTML` object

41. Examine the code below and justify what the answer is using pass by value and pass by reference:

```
let a = 'abc1';
```

```
let b = 'abc1';
console.log(a === b); // ?


let c = b;
b = 'abc1';
console.log(c); // ?


function function4(x){
  let d = 'abc100';
  console.log(d); //?
}


console.log(c); //->?
```

42. Describe how the JavaScript built-in functions **apply** and **call** work.

43. For the following object, use the call() method such that the string Welcome to Vancouver, BC in Canada is logged to the console:

```
let city = {
  name: 'Vancouver'
};
```

44. Have a look at the following object and answer the questions:

```
let superstore = {
  name: 'Kawaii Store',
  id: 1749,
  maxDiscount: 10,
  district: 'Tokyo',
  info: function(){
    console.log(this.name + ' ' + this.id + 'seasonal
discount is  $' + (this.maxDiscount *2))
```

```
  }
}
```

- Declare another object called `superStore2`. The value of its `name` property is `'Kool Things'`, the value of its `id` is `467` and the value of its `maxDiscount` property is 15
- There is no `info` key in the `superStore2` object. Call the `info` method from the `superStore` object such that the string `"Kool Things store id is 467 seasonal discount is $30"` is returned

45. Determine what the output of the following code will be:

```
var numberArr = [1, 89, 192, 10];
 (function () {
    console.log('Original array was: '  + numberArr);
    var numberArr = [13, 90];
    console.log('New array is: ' + numberArr);
})();
```

46. What will be returned from function `pizzaKing()` ?

47. The following returns an error. Modify the function with a helper so that the value of the `private` variable is accessible from outside of the function `secret()`:

```
function secret() {
  let private = 'The secret code is 18363xTk';
}
console.log(private); //ReferenceError: private is not
defined
```

48. Explain why the following condition is `false`:

```
0.1 + 0.2 === 0.3; //false
```

49. Code a function which will modify every alternate value of the following `fruit` array to uppercase letters:

```
let fruit = ['apple', 'pineapple', 'mango', 'berry'];
```

50. Code a function called `check(x)`. Within the body of the function check if the function has at-least one argument passed to it. If there are no arguments then return `false`, else return `true`. Use the following **try-catch** block to test function `check(x)`:

```
try {
  console.log(check());
} catch(err) {
  console.log(err);
}
```

51. Implement the `removeKey()` function, which takes an object name and key as parameters. If the object has the specified property the function should remove the property from the object and return the object. Else if the property does not exist in the object, then the function returns `false`

```
function removeKey(obj, property) {
}
```

52. Return the current date in the format Month/Date/Year

53. What is the value of the `result` variable?

```
let result = null ?? undefined ?? false;
```

54. Determine the values of arrays `b` and `c`:

```
let b = new Array(2);
console.log(b);
let c = new Array(-2);
console.log(c);
```

55. What does the debugger statement do?

56. Add the number 2 at the beginning of the `evenNumbers` array:

```
const evenNumbers = [4, 6, 8, 10, 12, 14, 16];
```

57. What is the difference between `splice()` and `slice()`

58. Using `splice()` add the `cities` array to the `country` array. It is ok to nest the `cities` array in the `country` array:

```
const cities = ['Toronto', 'Vancouver'];

const country=['Canada'];
```

59. Merge the contents of the `country` and `cities` array:

```
const cities = ['Toronto', 'Vancouver', 'Montreal',
'Victoria', 'Kamloops'];

const country=['Canada'];
```

60. Merge all the properties of the following objects:

```
const person = {
  name: 'Ree',
  position: 'Junior dev',
  salary: 100000,
};
const devTools ={
  OS: 'Mac',
  editor: 'Vim',
  browser: 'Chrome',
  headset: 'Oculus Quest',
};
const tasks = {
  client: ['Update code base', 'Debug form'],
  internal: 'Update bank information',
};
```

61. Merge the following two objects. What properties will the merged object have?

```
const furniture = {
  id: 1,
  type: 'Lawn table',
  available: true,
}
```

```
const appliances = {
  id: 100,
  type: 'Coffee maker',
  available: false
}
```

62. Explain what a callback function means, provide a simple example.
63. Join all the words in the following string:

```
let stringA = ' I like peace, tranquility and tacos';
```

64. Write a function which checks if two strings are anagrams of one another
65. What is the value of the `prime2` variable:

```
const prime1 = 5;
prime2 = prime1;
```

66. Write a function which returns the number of vowels in a word passed to it as an argument
67. What is a first class function in JavaScript?
68. What is the difference between `Object.create()` and `new Object`?
69. .Destructure the following array of objects:

```
let superArray =[
  {
    apple: 1,
    banana: 2
  },
  {
    cantaloupe: 3,
    dragonfruit: 4
  },
  {
```

```
    elderberry: 5,
    fig: 6
  }
]
```

70. Carrying on from #69, query the prototype of any of the three objects returned
71. Compare the two objects using ===. Explain your answer:

```
let object1 = {};
let object2 = {};
object1 === object2 //?
```

72. Write a function such that:

```
sum(1)(2);
```

73. JSON Pokémon data is available here: https://pokeapi.co/api/v2/pokemon Your job is to console.log/return only the names of the Pokémon. Hint Use the fetch API
74. For the following multi-dimensional array, use an array method such that the result is : `[100, 6023, 100, 1904, 800, 808, 1, 505]`

```
let multiDimensionalArray = [[100,6023], 100, [19045, 800], 808, [1, 505]];
```

75. This JSON list displays the oid, product name, supplier, quantity and unit cost property : value pairs of items.
    - Discount the `unit_cost` by 12% for each item `quantity` .
    - Add a `discount_price` property to each item whose value is the price after 12% off.
    - Push the `discount_price` and `supplier` properties to a `finalist` global array, such the array contains the following:

```
const finalList =["2404.75 Wisozk Inc", "2245.83 Keebler-Hilpert", "3812.01 Schmitt-Weissnat"];
```

```
let products = [{
  "_id": {
```

```
    "$oid": "5968dd23fc13ae04d9000001"
  },
  "product_name": "sildenafil citrate",
  "supplier": "Wisozk Inc",
  "quantity": 261,
  "unit_cost": "$10.47"
}, {
  "_id": {
    "$oid": "5968dd23fc13ae04d9000002"
  },
  "product_name": "Mountain Juniperus ashei",
  "supplier": "Keebler-Hilpert",
  "quantity": 292,
  "unit_cost": "$8.74"
}, {
  "_id": {
    "$oid": "5968dd23fc13ae04d9000003"
  },
  "product_name": "Dextromathorphan HBr",
  "supplier": "Schmitt-Weissnat",
  "quantity": 211,
  "unit_cost": "$20.53"
}]
```


76. What is the difference between an attribute and a property?

77. “All variable declarations are hoisted. Trying to access variables `let` and `const` variables before their declaration will throw an error, as they lie within the temporal dead zone.” - True or False?

78. For the following function constructor, answer the following questions:

```
function Mammal() {
  this.legs = 4;
  this.temperature = 'warm';
  this.ears = 2;
}
```

- Pass in 2 parameters (`animal` and `sound`) to the `Mammal` constructor function and modify the body of the function accordingly
- Create a new object `dog` from the `Mammal` function constructor passing in `'dog'` and `'woof'` as arguments
- Add a method called `greeting` to the dog object's prototype using the prototype property. The method will log to the console dog `'says woof'`. Keep in mind that you are added a method to the prototype object so you cannot hard code in the string `'dog says woof'`
- Call the `greeting` method on the dog object
- Query the prototype of the dog object

79. Convert the above `Mammal` function constructor into a class:

```
function Mammal() {
  this.legs = 4;
  this.temperature = 'warm';
  this.ears = 2;
}
```

- The constructor method of the class Mammal accepts two parameters animal and sound
- Add a instance/prototype method called greeting that will log to the console the name of the animal + 'says' and the sound. For example `'dog says woof'`
- Add a static method called `scientificName` which will return the string `'Mammalia'`
- Create a dog object using the `Mammal` class pass in `'dog'` and `'woof'` as arguments

- Access the `greeting` and `scientificName` methods on the `dog` object. What is returned?

80. Carrying on from question #79, create a `Dog` class which extends the `Mammal` class. It will have one constructor method. Add `animal`, `sound` and `breed` properties whose values are `'Dog'`, `'Woof Woof'` and the breed passed to an instance of the `Dog` class. Create a new `doggy1` instance from the `Dog` class and pass in the breed `'Boston Terrier'`
81. What does the **super** keyword do?
82. Classes are not hoisted and the body of a class is executed in strict mode. Is this statement true or false?
83. What is logged first and why?

```
setTimeout(callbackFunc, 3000);

function callbackFunc(){

  console.log('hello');

}

console.log('hey');
```

84. What is the event loop?
85. What is a higher-order function
86. Can you think of any array methods that follow the LIFO principle?
87. Write a recursive function?
88. Transform the array `nums`:

    Hint: *use map()*

```
//original array

let nums = [[1, 2], [3, 4]];


//transformed array

// nums = [[1, 4], [9, 16]];
```

89. For the following `courses` do the following:

- filter out the courses that are of the type `'Programming'` and less than `12` months in duration.
- From the filtered courses add the value of the `duration` property to check if the accumulated `duration` is `12` months:

```
courses = [
  {
    name: 'Programming in JavaScript',
    duration: 12,
    type: 'Programming'
  },
  {
    name: 'Responsive design',
    duration: 6,
    type: 'Design'
  },
  {
    name: 'Organic Chemistry',
    duration: 4,
    type: 'Science'
  },
  {
    name: 'Programming in Python',
    duration: 9,
    type: 'Programming'
  },
  {
    name: 'Physiology',
    duration: 9,
```

```
    type: 'Science'
  },
    {
    name: 'Introduction to Unity3D',
    duration: 3,
    type: 'Programming'
  },
];
```

90. What does callback hell refer to?

91. For the following HTML and array, use DOM methods to insert a list of the fruits and their ids into a webpage:

    HTML

    ```
    <div id ="fruits"></div>
    ```

    JavaScript

    ```
    const fruits = [
            {
                id: 1,
                type: 'Guavas'
            },
            {
                id: 2,
                type: 'Pineapple'
            },
            {
                id: 3,
                type: 'Apple'
            },
            {
    ```

```
            id: 4,
            type: 'Peach'
        },
        {
            id: 5,
            type: 'Dragon fruit'
        },
    ];
```

92. Have a look at the following block of code. The `setTimeout()` function will wait `0` seconds before executing its callback function. What is the order of the statements logged to the console? **Hint:** *event loop*

```
setTimeout(function(){
  console.log(100)
}, 0);
console.log(200);
console.log(300);
console.log(400);
console.log(500);
console.log(600);
console.log(700);
console.log(800);
```

93. What will be the output of `console.log(output)` statement?

```
let item = {
  type : 'Laptop'
};
let  output = (function(){
delete item.type;
return item.type;
```

```
}

)();

console.log(output);
```

94. What is the length of the array `flowers` after deleting flowers[1] ?

```
const flowers = ['Rosa', 'Lillium', 'Tulipa',
'Hyacinthus', 'Chrysanthemum', 'Gladiolus'];

delete flowers[1];
```

95. Make a deep clone of the following object:

```
const pizzas = {
   margherita: {
      toppings: ['spinach', 'mozzarella', 'pineapple'],
      prices: {
         small: '5.00',
         medium: '6.00',
         large: '7.00'
      },
     discount: null,
     date: new Date()
   },
   prosciutto: {
      toppings: ['peppers', 'mozzarella', 'olives'],
      prices: {
         small: '6.50',
         medium: '7.50',
         large: '8.50'
      },
     discount: undefined,
     date: new Date()
```

```
    }
};
```

96. Analyze the following statement. What will be logged to the console?

```
console.log(typeof(typeof(100)));
```

97. Use your knowledge of the event loop to analyze the following code and gauge what will be logged to the console:

```
console.log('Patience');

setTimeout(function(){console.log('Persistence');}, 1000);

console.log('Proficient');

console.clear();

setTimeout(function(){console.log('Peace');}, 0);
```

98. Consider the following object:

```
let laptop ={};
```

The `RAM` property is defined using the `Object.defineProperty()` method:

```
Object.defineProperty(laptop, 'RAM', {
  configurable: false,
  enumerable: false,
  writable: true,
  value: '16 GB'
});
```

- Write a getter method called getRAM to retrieve the value of the RAM property
- Using `Object.defineProperty` method, define a setter for the `laptop` object called `addedRAM`. This setter will assign a new value to the `RAM` property
- Assign `'32GB'` to the `addedRAM` setter

99. The following is some code for an airlines. The airlines gives 20% (`0.20`) bonus miles per round-trip as well `10` loyalty points per round trip. This is depicted by the

object airlines. The function `calculateBonusMiles(miles)` will return the sum of the `miles` and `loyalty_points` for a particular round trip multiplied by `0.20` (20% discount). For the following function below, complete the tasks:

```
let airMiles = {
 bonus_miles: 0.20,
 loyalty_points: 10
}
function calculateBonusMiles (miles) {
return( this.bonus_miles * (miles + this.loyalty_points))
}
```

- Declare variable `YVRtoKWT` which is a reference to the binding of the function `calculateBonusMiles(miles)` to the object `airMiles`
- Pass in the `14010` to `YVRtoKWT` to calculate the bonus air miles for one round trip

100. Calculate the Fibonacci series in JavaScript. This is a series of numbers in which each number (*Fibonacci number*) is the sum of the two preceding numbers. The simplest is the series 1, 1, 2, 3, 5, 8, etc.

Answers

1.
```
function countDown(num){
  if(num < 0) return
  console.log(num);  //recursive case
  countDown(num- 1)
}
countDown(10);
```

2. This will console log `5` as i is assigned the value of integer less than 5

3. Use `apply()`:

```
obj1.print = obj.print;
```

```
obj1.print.apply(obj1);//->7. Use apply on obj1 to print
7
```

4.

```
function isLeap(year) {
if (year % 4 == 0) {
   if( year%100!=0) {
     if(year%400==0){
     }
            console.log("is a leap year")

   }
  else {
  console.log('not')
}
}
} // end of function
isLeap(2104);
```

5. Use `let` or `var` instead of `const`

```
let myString = 'Hello 2020';
 myString = 'Hello World!';
 console.log(myString)//"Hello World!"
```

6. splice(index to start at, how many items to remove)

```
["King's Quest", "Super Mario", "Doom", "World of
Warcraft", "Zelda", "Contra"]
```

7. `textContent` can be used

8. `200` will be displayed

9. 5 and undefined, as the item at index 3 is no longer there. The array is now:

```
["JavaScript", "C#", "Rust", undefined, "Python"]
```

10.

```
arr1 =[];
```

11. The remaining values will be undefined:

```
[undefined, undefined, "Hello", undefined, undefined]
```

12. We can use the `onload()` method of the window object:

```
 window.onload = function(){
  console.log('hello world!');
}
```

13. True

14. The FIFO principle is used (First In First Out)

```
let myQueue.pop();
console.log(myQueue); //["a", "b"]
```

15. Use Object.assign() to create a shallow copy:

```
let keysCopy = Object.assign({}, keys);
```

16. Nothing will happen. The shallow copy remains unchanged.

17. In a deep copy, the source and target objects have different memory addresses and are not connected. Whereas, in a shallow copy the source and target objects share a memory address.

18.

```
let sortNumbers = function(nums) {
    for (let i = 0; i < nums.length; i++) {
        if ((nums[i] % 2) != 1) {
            even.push(nums[i]);
        }
        else {
            odd.push(nums[i]);
```

```
        }
    }
};
sortNumbers(nums);
console.log(even); // [2, 34, 54, 34, 32, 54, 66]
console.log(odd); // [1, 55, 11, 19, 17, 13]
```

19.

```
let i = 1;
do {
  if(i % 3 === 0 || i % 5 === 0){
    console.log(i);
  }
i++;
}
while (i < 101 );
```

20. True. As the statement `c` is not strictly equal to the boolean `false` is correct

21.

```
function octalSum(num1, num2){
  "use strict"
  num1 = num1.toString(8);
  num2 = num2.toString(8);
 let sum = num1 + num2;
  return(sum)
}
octalSum(10,10);
```

22.

```
function checkStrict(){
```

```
 'use strict'
 let x = 'test';
 delete x;
 console.log(x);
}
checkStrict();
```

23.

`console.log(b)` Logs to `2` as `b` is defined without the `"var"` keyword and is considered a global **var** which can be accessed outside of the function

`console.log(a)` Logs to an error as `a` is defined inside the function as a local **var**

24. For the following code, estimate what the console.log statements will log:
`console.log(person.details.print());` Prints `"Penelope Bruze"`

as this accesses the `name` property from the inner object that the `print` function is located in.

`console.log(detailsPrint());` Prints `"Jan Pan "` the global variable `detailsPrint` has access to the global variable `name`

25. Flatten the array using flat() and then use the `forEach()` method in conjuction with `indexOf()` to remove an item that occurs more than once in the array

```
let num1 = [1, 2, 3, 4];
let num2 = [10, 20, 30, 1];
let num3 = [100, 200, 300, 1];
let mergedArray = [num1, num2, num3].flat();
let result = mergedArray.filter(function(item, position)
{
    return mergedArray.indexOf(item) == position;
});
console.log(result); //[1, 2, 3, 4, 10, 20, 30, 100, 200,
300]
```

26. `false`. The value of the global variable `classroom` is logged

27. `false`. Symbol() are unique identifiers, regardless of the value assigned.

28.

```
nums.forEach(function(item, index) {
    console.log(index); // 0, 1, 2
    item = item * 2;
    newNums.push(item);
});
console.log(newNums); // [2, 4, 6]
```

29. A property of an object can also be a function which is referred to as a method. Within a method **this** refers to the object within which it is based on.

```
let dog = {
  name: 'Rover',
  breed: 'Boxer',
  treat: 'Cookies',
  callDog: function(){
    console.log(this.name);
  }


}
dog.callDog(); //"Rover"
```

30. Arrow functions do not have a **this** binding.

31. `true`. A function assignment is not strictly equal to an object.

32. The pre-increment operator (++x) will add one to the variable and return it. Whereas, the post-increment operator (x++) will return the original value of the variable before incrementing it. The following will be logged to the console:

```
32
30
32
```

33. Function to generate a random number between 1-100:

```
function random(){
   console.log(Math.floor(Math.random() * 101));
  // you can also use return
}
random();
```

34. Function to return a random array item:

```
function random(x){
let randomItem = x[Math.floor(Math.random() * x.length)];
  return(randomItem);
}
console.log(random(items));
```

35. Function to find if a given number is to the power of 2

```
function isPower(num){
  if(num % 2 === 0){
  console.log(num + " is a power of 2")
  }
  else{
 console.log(num + " is not a power of 2")

  }
} // end of function

isPower(44); //"44 is a power of 2"
```

36. 3 different ways to empty an array:

```
let alphas =['a', 'b', 'c', 'e', 'k'];
 1. alphas = [];
 2. alphas.length = 0;
 3. while(alphas.length){
      alphas.pop();
    }
 4. alphas.splice(0, alphas.length);// start from 0 till
the length of the array
```

37. `array2` shares the same reference address as `array1`. Therefore, modifying `array1` will also modify `array2`

```
let array1= ['a','b','c','d','e','f']; // Created array
let array2 = array1;
array1.push('k');
console.log(array2);// ["a", "b", "c", "d", "e", "f", "k"]
console.log(arra2.length); // 7
```

38. Function to mask an email:

```
let email = 'abcde12345@gmail.com';

function maskEmail(email) {
 let splitEmail = email.split("@");
 let userEmail = (splitEmail[0]);
 let domain = (splitEmail[1]);
 if (userEmail.length <= 5) {
  let userEmailSplit = userEmail.split("");
  let firstLetter = userEmailSplit[0];
  let sliceUserEmail = userEmailSplit.slice(1);
```

```
  let i = 0;
  for (i = 0; i < sliceUserEmail.length; i++) {
   sliceUserEmail[i] = "."
   var joinUserEmail = sliceUserEmail.join("");

  } // end of for loop
  let maskedEmail = firstLetter + joinUserEmail + "@" +
domain;
 } // end of if block
 else {
  let userEmailSplit = userEmail.split("");
  let firstFiveLetters = userEmailSplit.slice(0, 5);
  let joinedName = firstFiveLetters.join("");
  let restUserName = userEmailSplit.slice(5,
userEmailSplit.length);
  // for loop

  let i = 0;
  for (i = 0; i < restUserName.length; i++) {
   restUserName[i] = "."
   joinedRestUserName = restUserName.join("");
  } // end of for loop
  let maskedEmail = joinedName + joinedRestUserName + "@"
+ domain;
  console.log(maskedEmail);

 } // end of else block
```

```
} // end of function

maskEmail(email)
```

39. `course` function constructor:

```
function course(duration, students, instructor){

  this.duration = duration;

  this.students = students;

  this.instructor= instructor;

}

let IntroductionToHtml = new course("1 month", "31",
"Miss. Boo");

let IntroductionToCss = new course("1 month", "100", "Mr.
Ray");

let IntroductionToSass = new course("1 month", "150",
"Mrs. KayBee");


console.log(IntroductionToHtml);

console.log(IntroductionToCss);

console.log(IntroductionToSass);
```

40.

```
delete IntroductionToHtml.duration;
```

41.

```
let a = 'abc1';

let b = 'abc1';

console.log(a===b); // ? // This console logs to true.

//It is a comparison operation by Value. Both the string
values of a and b are the same


let c = b;

b = '1';
```

```
console.log(c); // This will console log to 'abc1'.

/*The value of  b is assigned to c. And then b is
assigned a new value of '1'.

This new value is assigned a new location in memory,
whereas c still refers to the location in memory of let b
which was assigned to 'str1'.

This an example or reference by

*/


function function4(x){

  let d = 'abc100';

  console.log(d); //?

}

  function4(c); // logs the value of local variable d
which is "abc100"


console.log(c); //->?   logs "abc1".

/*As the value of b was assigned to c.

 b at the time of assignment contained the value of
'abc1'. The value is by reference.

As c still refers to the location in memory of  b which
was assigned to 'abc1' */
```

42. Apply: This is a built in JavaScript method which changes **"this"** to refer to a new object. It is convenient for applying the function of one object to another without having to replicate the function within the new object.

    Call: Like apply, call changes **this** to refer to a new object. The difference between **apply** and **call** is that an array may be passed to **apply** as a parameter whereas call will accept the parameter as is.

43.

```
let greeting = function(a, b){
```

```
 return 'Welcome to ' + this.name + ', ' + a + ' in ' + b
};
console.log(greeting.call(city, 'BC', 'Canada'));
// "Welcome to Vancouver,BC in Canada"
```

44. Use the **apply()** method to apply an object's method on another object:

```
let superstore = {
  name: 'Kawaii Store',
  id: 1749,
  maxDiscount: 10,
  district: 'Tokyo',
  info: function(){
    return(this.name + ' store id is ' + this.id + '
seasonal discount is  $' + (this.maxDiscount *2));
  }
}
let superStore2 ={
  name: 'Kool Things',
  id: 467,
  maxDiscount: 15
}
console.log(superstore.info.apply(superStore2));
```

45. The output is:

```
"Original array was: undefined"
"New array is: 13,90"
```

The `numberArr` within the `IIFE` is hoisted to the top of the function, where it is initialized with a value of `undefined`. To make this clearer consider the following example:

```
console.log(x);

var x = 5; //undefined
```

46. An error is logged as function expressions are not hoisted:

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

This is what the process looks like:

```
let pizzaKing; //undefined

pizzaKing() // undefined


pizzaKing = function() {
 return 100
}
```

47. Create a helper function within the outer `secret()` function which will return the value of the variable `private`. The assign the return value to another variable declared as `privateValue`:

```
function secret() {
  let private = 'The secret code is 18363xTk';
  return function(){
    return private;
  }
}
let privateValue = secret();

console.log(privateValue()); //"The secret code is
18363xTk"
```

48. `false` is returned due to floating point errors. Computers use a format (binary floating-point) that cannot accurately represent numbers such as 0.3, 0.1 etc. For more information go to: https://0.30000000000000004.com/

49.

```
let fruit = ['apple', 'pineapple', 'mango', 'berry'];
let modifiedFruit = [];
 fruit.forEach(function(item, index){
    if(index % 2 !== 0) {
       modifiedFruit.push(item.toUpperCase())
    }
});
console.log(modifiedFruit); ["PINEAPPLE", "BERRY"]
```

50. Use the `arguments` object:

```
function check(x) {
  if(arguments.length <= 0){
    return false;
  } else {
    return true;
  }
}
```

51. Check the function with an example `item` object:

```
let item = {
 id: 1020,
 type: 'Standing table',
 price: '$99.99',
}
function removeKey(obj, property) {
 for (let key in obj) {
  if (obj[property]) {
   delete obj[property]
```

```
   console.log(obj)
  }
 }
}
removeKey(item, 'id');
console.log(item);
```

```
Object {
  price: "$99.99",
  type: "Standing table"
}
```

52. Use the `date` object along with the `toLocaleDateString()` method to convert the `date` object into a readable string:

```
let date = new Date();
let formattedDate = date.toLocaleDateString();
console.log(formattedDate); //"7/27/2020"
```

53. `false`, as falsy values are accepted by the nullish coalescing operator.

54.

```
let b = new Array(2); // intialize an array with a length of 2
console.log(b); //? [undefined, undefined]
let c = new Array(-2); // negative number is out of range to intialize an array
console.log(c); // RangeError: invalid array length
```

55. The debugger statement stops the execution of JavaScript. If a debugger function is available it will call it. It adds breakpoints in your code so that you can debug effectively.

56. Use the `unshift()` method:

```
evenNumbers.unshift(2); // [2, 4, 6, 8, 10, 12, 14, 16]
```

57. `slice()` extracts a section of a string and returns the selected items in a new array. It does not change the original array. The `splice()` method adds/removes items from an array, and returns the removed items. This method mutates the original array.

58. Start at index `0`, remove `0` items and add the `cities` array:

```
country.splice(1, 0, cities); // ["Canada", ["Toronto", "Vancouver"]]
```

59. Use the spread operator to merge the contents of the array:

```
const merged = [...country, ...cities];

// ["Canada", "Toronto", "Vancouver", "Montreal", "Victoria", "Kamloops"]
```

60. Merge the objects using the spread operator:

```
const mergedObjects = {...person, ...devTools, ...tasks};
```

61. When objects have the same property names, the last object's properties will be present in the merged object:

```
const furniture = {
  id: 1,
  type: 'Lawn table',
  available: true,
}
const appliances = {
  id: 100,
  type: 'Coffee maker',
  available: false,
}
let mergedItems = {...furniture, ... appliances};
console.log(mergedItems);
```

```
[object Object] {
  available: false,
  id: 100,
  type: "Coffee maker"
}
```

62. A callback function is a function that is passed to another function as an argument. It is executed after some operation has been completed:

```
let greetingMessage = function() {

    console.log('This message will be shown after 10 seconds');

}


setTimeout(greetingMessage, 10000);
```

63. Use `split()` using white space `' '` as a seperator to return an array of characters and then and `join()` to join the characters. `join()` method combines the elements in an array and returns it as a string separated by a comma or any specified string separator such as `''`.

```
let stringB = (stringA.split(' ').join(''));

// "Ilikepeace,tranquilityandtacos"
```

64.

```
function anagram(string1, string2) {

  // For case insensitivity, change both words to lowercase.

  var a = string1.toLowerCase();

  var b = string1.toLowerCase();


  // Sort the strings, and join the resulting array to a string

  a = a.split('').sort().join('');
```

```
  //console.log(a) is ["e", "i", "l", "n", "s", "t"] with
split()

  // and "eilnst" with join()

  b = b.split('').sort().join('');

  return(a === b);

}

anagram('LISTEN', 'silent'); // true
```

65. `prime2` is not declared with either **var**, **let** or **const** so it will be declared in the global namespace with `var`

```
console.log(prime2); // 5
```

66.

```
function countVowels(word) {

  let count = 0;

  let vowels = ['a', 'e', 'i', 'o', 'u'];

  let lowerCaseWord = word.toLowerCase();


  //loop through the word to check if each character is a
vowel

  for (let letter of lowerCaseWord) {

    console.log(letter)

    if (vowels.includes(letter)) {

      count++

    }

  }


  //return count

  console.log(count)

}
```

```
countVowels('love'); //2
```

67. This means that functions are treated like any other variable. For example, ,a function can be passed as an argument to other functions, can be returned by another function and can be assigned as a value to a variable.

68. `Object.create()` method creates a new object, using an existing object as the prototype of the newly created object. `new Object` used the built in constructor function to create a new object.

69. Destructuring arrays:

```
let[one, two, three] = superArray;
console.log(one);
/*object Object] {
  apple: 1,
  banana: 2
*/
```

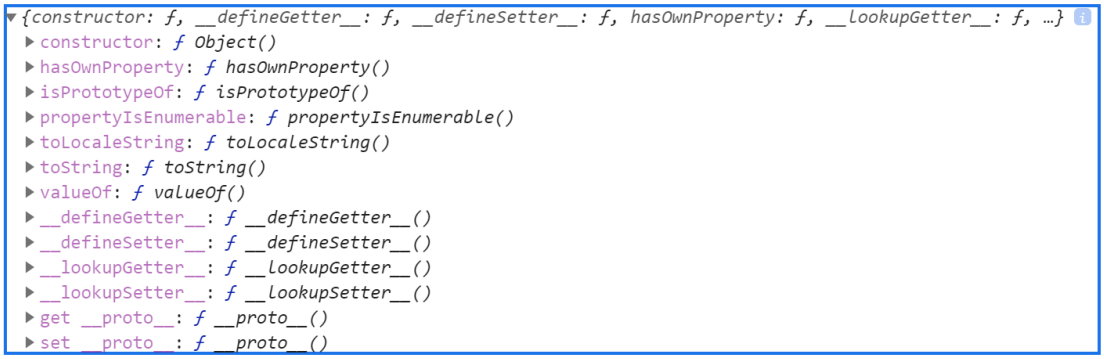
70. Use the `Object.getPrototype()` method which will return the built in :

```
let superArray =[
  {
    apple: 1,
    banana: 2
  },
  {
    cantaloupe: 3,
    dragonfruit: 4
  },
  {
    elderberry: 5,
    fig: 6
  }
```

```
]


let  [one, two, three] = superArray;

console.log(Object.getPrototypeOf(one));
```



71. When comparing two objects === will check if the two objects reference the same location in memory. Which they donot:

```
console.log(object1 === object2 );// false
```

72. Use a closure:

```
function sum(a){
  return function(b){
    console.log(a + b)
  }
}
```

73. https://pokeapi.co/api/v2/pokemon/

```
fetch("https://pokeapi.co/api/v2/pokemon/")
  .then(results => {
    return results.json();
  }).then(data => {
    let items = data.results
    items.forEach(function(item) {
      console.log(item.name)
```

```
    })

  })
```

74. Use the `flat()` method passing in `Infinity` as an argument:

```
let flattenedArray =multiDimensionalArray.flat(Infinity);
```

75. JSON practice question:

```
const finalList = [];

products.forEach(function(item) {

  let price = item.unit_cost;

  //remove the $ sign

  let unit_price = parseFloat(price.slice(1));

  let quantity = item.quantity;

  let total_cost = (quantity * unit_price) - (quantity *
unit_price * 0.12)

  item.discount_price = total_cost.toFixed(2);

  finalList.push(`${item.discount_price}
${item.supplier}`)

});

console.log(finalList);
```

```
["2404.75 Wisozk Inc", "2245.83 Keebler-Hilpert",
"3812.01 Schmitt-Weissnat"]
```

76. Attributes are defined by HTML. These provide additional information about an element. For example href, class, id, value are some HTML attributes. Properties are defined by the DOM. Upon parsing a document, the browser creates a hierarchical representation of all the elements in a webpage. These are called node objects and have properties such as className, children associated with them.

77. True.

78. `Mammal()` function constructor:

```
function Mammal(animal, sound){
```

```
  this.legs = 4;
  this.temperature = 'warm';
  this.ears = 2;
  this.animal = animal;
  this.sound = sound;
}
let dog = new Mammal('dog', 'woof');

Mammal.prototype.greeting = function(){
  console.log(`${this.animal} says ${this.sound}`);
}
dog.greeting(); //"dog says woof"
console.log(Object.getPrototypeOf(dog)); // Mammal
```

79. `Mammal` class

```
class Mammal{
  constructor(animal, sound){
  this.legs = 4;
  this.temperature = 'warm';
  this.ears = 2;
  this.animal = animal;
  this.sound = sound;
  }
  greeting(animal, sound){
    console.log(`${this.animal} says ${this.sound}`);
  }
  static scientificName(){
    console.log('Mammalia')
```

```
  }
}
let dog = new Mammal('dog', 'woof');
dog.greeting();//"dog says woof"
dog.scientificName(); // error
```

80.

```
class Mammal{
  constructor(){
  this.legs = 4;
  this.temperature = 'warm';
  this.ears = 2;
  }
  static scientificName(){
    console.log('Mammalia')
  }
}

class Dog extends Mammal{
  constructor(breed){
    super()
    this.animal = 'Dog';
    this.sound = 'Woof Woof';
    this.breed = breed
  }
}

  let doggy1 = new Dog('Boston Terrier');
```

```
console.log(doggy1);
```

81. The **super** keyword is used to call the constructor of the parent class and to access the parent's properties and methods.

82. True.

83. The following is the order of messages logged :

```
"hey"
"hello"
```

    This is an example of an asynchronous callback function. `callbackFunc()` will be called 3 seconds after, in the meantime the global `console.log` statement will execute.

84. The event loop monitors the call stack and event queue. When the call stack is empty, it will take an event (a callback function) from the event queue and push it to the call stack. Then the call stack executes the event/function and pops it off the call stack. Each such cycle is called a tick.

85. A function that accepts another function as an argument or returns a functionis called a higher-order function

86. The array push and pop methods follows the LIFO principle. When an element is pushed into an array, it is added towards the end:

```
let array1 = [1,2,3];
array1.push(4);
console.log(array1); // [1, 2, 3, 4]
```

    Using the `pop()` method the last item to be added into an array is removed (first out)

```
array1.pop();
console.log(array1); // [1, 2, 3]
```

87. Recursive function with no base case to exit the loop, will cause the stack to overflow:

```
function recursion(){
  recursion();
```

```
}

recursion();
```

88. Flatten the array, use the `map()` method and then re-create a 2D array:

```
let nums = [[1,2], [3,4]].flat();

//mutating the original array

const mappedNums = nums.map(function(item){

  return item * item

})

//empty the nums array to reset it

nums = [];

nums.push([mappedNums[0], mappedNums[1]],[mappedNums[2],
mappedNums[3]]);

console.log(nums); // [[1, 4], [9, 16]]

// without mutating the original array

let nums1 = nums.map( num => num.map(n => n*n))

console.log(nums1); [[1, 4], [9, 16]]
```

89.

```
let programming = courses.filter(function(item) {

      return item.type === 'Programming' && item.duration
< 12

}).map(function(item) {

      return item.duration

}).reduce(function(acc, item) {

      return acc + item

}, 0)

console.log(programming); //12
```

90. Multiple nested callback functions are known as callback hell.

91. Iterate over the array of fruits and output the results to the DOM in an `<ul></ul>` tag

```
let fruitDiv = document.getElementById("fruits");
fruitDiv.innerHTML = "<h1>Fruit List</h1>";
for(var x = 0; x < fruits.length; x++) {
     fruitDiv.innerHTML += "<ul id='myUL'><li>" +
fruits[x].id + ": " + fruits[x].type + "</i></ul>";
}
```

Fruit List

- 1: Guavas
- 2: Pineapple
- 3: Apple
- 4: Peach
- 5: Dragon fruit

92. The callback function passed to `setTimeout()` will execute at the end once the call stack is empty

```
200
300
400
500
600
700
800
100
```

93. `undefined`. As we have deleted the `type` property of the global `item` object from within the self-invoking function

94. The `length` of the array `flowers` after deleting `flowers[1]` is `6`. As deleting an array item does not alter its `length` property. `flowers[1]` now has a value of `undefined`.

```
console.log(flowers.length);

console.log(flowers);
```

```
6

["Rosa", undefined, "Tulipa", "Hyacinthus",
"Chrysanthemum", "Gladiolus"]
```

95. Deep clone the object with a custom function as it has a property with a value of `null`, `date` and `undefined`:

```
function cloneObject(obj) {
    let clone = {};
    for(let i in obj) {
        if(obj[i] != null &&  typeof(obj[i]) ===
"object")
            clone[i] = cloneObject(obj[i]);
        else
            clone[i] = obj[i];
    }
    return clone;
}
let pizzas_copy = cloneObject(pizzas);
```

96. `"string"` will be logged as `typeof(100)` returns `"number"` and `typeof("number")` returns `"string"`

97. The following is logged to the console:

```
"Peace"

"Persistence"
```

`console.log('Patience')` and `console.log('Proficient')` are synchronous and logged first but `console.clear()` clears them from the console. After which the `setTimeout()` methods execute once the call stack is empty

98. `getDetails()` getter method:

```
Object.defineProperty(laptop, 'getDetails', {
  get: function() {
    console.log(this.RAM);
  }
});
```

`addedRAM` setter method:

```
Object.defineProperty(laptop, 'addedRAM',{
  set: function(addedRAM){
    this.RAM = addedRAM
  }
});
laptop.addedRAM = '32GB';
console.log(laptop.RAM); // "32GB"
```

99. use the `bind()` method to bind the **this** context of the object to the function:

```
let YVRtoKWT = calculateBonusMiles.bind(airMiles);
YVRtoKWT(14010); //2804
```

100. Fibonacci series:

```
let fibonacciSeries = function (num)

{
  if (num === 1){
    return [0, 1];
```

```
  } else{
    let a = fibonacciSeries(num - 1);
    a.push(a[a.length - 1] + a[a.length - 2]);
    return a;
  }
};
console.log(fibonacciSeries(10));
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```