const heroes = [

{ name: 'Wolverine', family: 'Marvel', isEvil: false },

{ name: 'Deadpool', family: 'Marvel', isEvil: false },

{ name: 'Magneto', family: 'Marvel', isEvil: true },

{ name: 'Charles Xavier', family: 'Marvel', isEvil: false },

{ name: 'Batman', family: 'DC Comics', isEvil: false },

{ name: 'Harley Quinn', family: 'DC Comics', isEvil: true },

{ name: 'Legolas', family: 'Tolkien', isEvil: false },

{ name: 'Gandalf', family: 'Tolkien', isEvil: false },

{ name: 'Saruman', family: 'Tolkien', isEvil: true }

]

// 1. How to preserve the immutability on my heroes list? Solve below problems using the same

//Spread operator ... can be used to preserve the immutability of heroes list.

// a. Get heroes who are not evils

let heroesNotEvils = heroes.filter(hero=>!hero.isEvil)

console.log("a. Get heroes who are not evils...", heroesNotEvils)

// b. Print Unique family names

let uniqueFamilyNames = heroes.reduce((prevVal, currVal, index, array) => {

prevVal[currVal.family] = prevVal[currVal.family] ? prevVal[currVal.family] + 1 : 1

return prevVal

}, [])

console.log("b. Print Unique family names...", uniqueFamilyNames)

// c. Print Hero Names from given objects, and append sir in each of them before printing

let heroesWithSirInNames = heroes.map(hero=> hero.name + " Sir")

console.log("c. Print Hero Names from given objects, and append sir in each of them before printing...", heroesWithSirInNames)

// d. Do we have any hero in Marvel Family who is not evil

let heroWhoIsNotEvil = heroes.some(hero => !hero.isEvil)

console.log("// d. Do we have any hero in Marvel Family who is not evil...", (heroWhoIsNotEvil == true ? "YES" : "NO"))

// const heroes = [

// { name: 'Wolverine', family: 'Marvel', isEvil: false },

// { name: 'Deadpool', family: 'Marvel', isEvil: false },

// { name: 'Magneto', family: 'Marvel', isEvil: true },

// { name: 'Charles Xavier', family: 'Marvel', isEvil: false },

// { name: 'Batman', family: 'DC Comics', isEvil: false },

// { name: 'Harley Quinn', family: 'DC Comics', isEvil: true },

// { name: 'Legolas', family: 'Tolkien', isEvil: false },

// { name: 'Gandalf', family: 'Tolkien', isEvil: false },

// { name: 'Saruman', family: 'Tolkien', isEvil: true }

// ]

//2. Use the spread and rest operator to create a function which can multiple numbers from 1...n (n is the number of choice)

// also need to print students of the session using same example

let multiplyNumbers = (...numbers) => {

let result = 1;

for(let i in numbers) {

result = result \* numbers[i]

}

return result

}

console.log("2. Use the spread and rest operator to multiply numbers...", multiplyNumbers(1,2,3))

//3. Print the last name through destructuring and add a contact number:9119119110 as well

const person = {

userDetails :{

first: "FirstName",

last: "LastName"

}

}

let{ userDetails : {last, contactNumber = 9119119110}} = person

console.log("3. Print the last name through destructuring..", last, " and Contact Number is ...", contactNumber)

//4. Give me an example of const data manipulation

const number = 10;

//number = 20;----------------re-assignment to const variable not allowed

//console.log(number)

const Employee1 = {

name : "Employee1",

designation : "Programmer"

}

const Employee2 = {

name : "Employee2",

designation : "DBA"

}

//Employee1 = Employee2 //assigning reference to const is not allowed

Employee1.name = "Changed Employee1 name" //updating value of reference allowed

console.log(Employee1) //prints { name: 'Changed Employee1 name', designation: 'Programmer' }

//5. What is the difference between for-of and for-in show with examples

/\*for-of : It returns values of object iterated over and is used to iterate over property names and used for arrays. Basically, array of numbers or strings instead of objects.

for-in : It returns keys of object iterated over and is used to iterate over property values. It is mostly used with json objects with key-values.

\*/

//Example of for-in

let product = {name : "cart1", price: 100, items : {}}

for(const key in product) {

console.log("5. for-in example: ", key)

}

let products = ["Product1", "Product2", "Product3"]

for(const product of products) {

console.log("5. for-of example: ", product)

}

//6. Give me an example of bind and write its usage, comparison with arrow function

//Function bind is used in when we want block exceution of piece of code for certain time period.

var Employee = {

name : "Employee1",

salary: "3000",

GetEmployeeInfo : function() {

setTimeout((function() {

console.log(`6. employee name using bind: ${this.name}`) //it shows employee name: Employee1 as context with Employee is bound using bind function

}).bind(Employee), 10)

}

}

Employee.GetEmployeeInfo()

//Arrow function copies the context of immediate parent function

var Directory = {

name : "Person1",

phone: "777-999-0099",

GetDirectoryInfo : function() {

setTimeout (() => {

console.log(`name using Arrow function: ${this.name}`) //it shows name: Person1 as context with Directory is bound using arrow function

}, 10)

}

}

Directory.GetDirectoryInfo()

//7. Create an example showing usage of event loop in concurrent execution cycle

//event loop is used to allow non-blocking operations and create queue for the tasks that needs to be exceuted aftre some delay

//test function in below example is not called immediately after printing "Function starts here..."

function sayHi() {

console.log("Function starts here...")

setTimeout(test, 0)

console.log("Function ends here...")

}

function test() {

console.log("Waiting to be exceuted...")

}

sayHi()

//8. create an example showing usage of short hand and default param.

//if key and value refer to same variable, short hand method can be used

let country1 = "USA",

country2 = "India",

country3 = "UK",

country4 = "SA"

let Countries = {country1, country2, country3, country4}

console.log("8. create an example showing usage of short hand...", Countries)

//9. Create two objects with some properties and merge them using Object method and ES6 way

//Using Object method

var Employee = {

name : "Employee1",

designation : "Developer",

salary : 5000

}

var Department = {

deptName : "Development",

deptCode : 30

}

var employeeDept = Object.assign(Employee, Department)

console.log("9. Object.assign to merge objects...", employeeDept)

//ES6 way

let EmployeeES6 = {name : "Employee1", designation : "Developer", salary : 5000}

let DepartmentES6 = {deptName : "Development", deptCode : 30}

let employeeDeptES6 = Object.assign(EmployeeES6, DepartmentES6)

console.log("9. ES6 way to merge objects...", employeeDeptES6)

//10. Give me an example of call and apply each with it's usage

//call is a function that helps change the context of the invoking function by passing an object

//apply is similar to call except it accepts an array as an argument

var Student ={name: "Student1", ID: 1, course: "English"}

var courses = ["Maths", "Social studies", "Spanish"]

console.log("10. Example of call... ")

function printStock(course1, course2, course3) {

console.log(`Student details- ${this.name}, ${this.ID}, ${this.course}`)

console.log(`${course1}, ${course2}, ${course3}`)

}

printStock.call(Student, "Computer programming", "Business", "Arts")

printStock.apply(Student, courses)