# Solutons-Exercises: Advanced Functions

## 1.Sort Array

function sortArray(inputArray, sortOrder){

// if(sortOrder === 'asc'){

// return inputArray.sort((a, b) => a - b);

// } else if(sortOrder === 'desc'){

// return inputArray.sort((a, b) => b - a);

// }

let ascendingComparator = function (a, b){

return a - b;

}

let descendingComparator = function (a, b){

return b - a;

}

let sortingStrategies = {

'asc': ascendingComparator,

'desc': descendingComparator

}

return inputArray.sort(sortingStrategies[sortOrder]);

}

console.log(sortArray([14, 7, 17, 6, 8], 'asc'))

console.log(sortArray([14, 7, 17, 6, 8], 'desc'))

|  |
| --- |
| function solve(arr, sortOrder) { |
|  | return arr.map(Number).sort((a, b) => (sortOrder === "asc" ? a - b : b - a)); |
|  | } |

|  |
| --- |
| let sort = (arr, sortMethod) => arr.sort((a, b) => sortMethod === 'desc' |
|  | ? b - a |
|  | : a - b); |
|  |  |
|  | console.log(sort([14, 7, 17, 6, 8], 'asc')); |

## 2.Argument Info

function getArgumentsInfo(){

let typeCounts = {};

for (let i = 0; i < arguments.length; i++) {

let argument = arguments[i];

let type = typeof arguments[i];

console.log(type + ': ' + argument);

if(!typeCounts[type]){

typeCounts[type] = 0;

}

typeCounts[type]++;

}

let sortedTypeCounts = [];

for(let type in typeCounts){

sortedTypeCounts.push([type, typeCounts[type]]);

}

sortedTypeCounts.sort((a, b) => b[1] - a[1]);

for(let typeAndCount of sortedTypeCounts){

console.log(typeAndCount[0] + ' = ' + typeAndCount[1]);

}

}

getArgumentsInfo('cat', 42, function () { console.log('Hello world!'); }, 19)

|  |
| --- |
| function argumentsInfo() { |
|  | let metaData = new Map(); |
|  |  |
|  | for (const arg of arguments) { |
|  | let currentType = typeof arg; |
|  | console.log(`${currentType}: ${arg}`); |
|  |  |
|  | let test = metaData.get(currentType) |
|  | if (metaData.get(currentType)) { |
|  | metaData.set(currentType, metaData.get(currentType) + 1); |
|  | } else { |
|  | metaData.set(currentType, 1); |
|  | } |
|  | } |
|  |  |
|  | [...metaData] |
|  | .sort((a, b) => b[1] - a[1]) |
|  | .forEach(md => { |
|  | console.log(`${md[0]} = ${md[1]}`); |
|  | }); |
|  | } |
|  |  |
|  | argumentsInfo('cat', 42, function () { console.log('Hello world!'); }); |

|  |
| --- |
| function solve() { |
|  | let counter = new Map(); |
|  | for (let i = 0; i < arguments.length; i++) { |
|  | const ar = arguments[i]; |
|  | console.log(`${typeof ar}: ${ar}`) |
|  | if (!counter.has(typeof ar)) { |
|  | counter.set(typeof ar, 0); |
|  | } |
|  | counter.set(typeof ar, counter.get(typeof ar) + 1); |
|  | } |
|  | Array.from(counter.entries()).sort((a, b) => b[1] - a[1]).forEach(entry => { |
|  | console.log(`${entry[0]} = ${entry[1]}`); |
|  | }); |
|  | } |
|  |  |
|  | solve({ |
|  | name: 'bob' |
|  | }, 3.333, 9.999) |

## 3.Functional Sum

function add(number){

let sum = number;

function sumNumbers(num){

sum += num;

return sumNumbers;

}

sumNumbers.toString = function (){

return sum;

};

return sumNumbers;

}

console.log(add(1).toString())

console.log(add(1)(6)(-3).toString())

|  |
| --- |
| let add = (function () { |
|  | let sum = 0; |
|  |  |
|  | function result(num) { |
|  | sum += num; |
|  | return result; |
|  | } |
|  |  |
|  | result.toString = () => sum.toString(); |
|  |  |
|  | return result |
|  | })(); |
|  |  |
|  | console.log(add(1).toString()); |
|  | console.log(add(1)(6)(-3).toString()); |

|  |
| --- |
| function add(num) { |
|  | let sum = num; |
|  |  |
|  | function calc(num2) { |
|  | sum += num2; |
|  | return calc; |
|  | } |
|  |  |
|  | calc.toString = function () { |
|  | return sum |
|  | }; |
|  | return calc; |
|  | } |
|  | console.log(add(5)(5)(5)) |

## 4.Personal BMI

function getPersonalBMI(name, age, weight, height){

let bmi = Math.round(weight / (height / 100) / (height / 100));

let personalBMI = {

name: name,

personalInfo: {

age: age,

weight: weight,

height: height

},

BMI: bmi

};

let status = '';

if(bmi < 18.5){

status = 'underweight';

} else if(bmi < 25){

status = 'normal';

} else if(bmi < 30){

status = 'overweight';

} else if(bmi >= 30){

status = 'obese';

}

personalBMI.status = status;

if(status === 'obese'){

personalBMI.recommendation = 'admission required';

}

return personalBMI;

}

console.log(getPersonalBMI('Peter', 29, 75, 182))

console.log(getPersonalBMI('Honey Boo Boo', 9, 57, 137))

|  |
| --- |
| function personalBMI(name, age, weight, height) { |
|  | let bmi = Math.round(weight / (height / 100) / (height / 100)); |
|  |  |
|  | let result = { |
|  | name: name, |
|  | personalInfo: { |
|  | age: age, |
|  | weight: weight, |
|  | height: height |
|  | }, |
|  | BMI: bmi |
|  | } |
|  |  |
|  | let status = (result.BMI < 18.5) ? 'underweight' |
|  | : (result.BMI < 25) ? 'normal' |
|  | : (result.BMI < 30) ? 'overweight' |
|  | : 'obese'; |
|  |  |
|  | result.status = status; |
|  | if (result.BMI >= 30) { |
|  | result.recommendation = 'admission required'; |
|  | } |
|  |  |
|  | return result; |
|  | } |
|  |  |
|  | console.log(personalBMI('Peter', 29, 75, 182)); |

|  |
| --- |
| function solve(name, age, weight, height) { |
|  | age = Number(age); |
|  | weight = Number(weight); |
|  | height = Number(height); |
|  |  |
|  | let result = {}; |
|  | result.name = name; |
|  | result.personalInfo = { |
|  | age: age, |
|  | weight: weight, |
|  | height: height |
|  | }; |
|  | result.BMI = Math.round(weight / (height \* height / 10000)) |
|  |  |
|  | if (result.BMI < 18.5) { |
|  | result.status = "underweight"; |
|  | } else if (result.BMI < 25) { |
|  | result.status = "normal"; |
|  | } else if (result.BMI < 30) { |
|  | result.status = "overweight"; |
|  | } else if (result.BMI >= 30) { |
|  | result.status = "obese"; |
|  | result.recommendation = "admission required"; |
|  | } |
|  |  |
|  | return result; |
|  | } |
|  |  |
|  | console.log(solve("Peter", 29, 75, 182)); |

## 5.Vector Math

//In Judge must be paste only the function without variable

let vectorProduct = (function() {

function add(firstVector, secondVector){

let addedVector = [];

addedVector[0] = firstVector[0] + secondVector[0];

addedVector[1] = firstVector[1] + secondVector[1];

return addedVector;

}

function multiply(vector, scalar){

let scalaredVector = [vector[0] \* scalar, vector[1] \* scalar];

return scalaredVector;

}

function length(vector){

let vectorLength = Math.sqrt(vector[0] \* vector[0] + vector[1] \* vector[1])

return vectorLength;

}

function dot(firstVector, secondVector){

let vectorsDotProduct = firstVector[0] \* secondVector[0] + firstVector[1] \* secondVector[1];

return vectorsDotProduct;

}

function cross(firstVector, secondVector){

let vectorsCrossProduct = firstVector[0] \* secondVector[1] - firstVector[1] \* secondVector[0];

return vectorsCrossProduct;

}

return {

add: add,

multiply: multiply,

length: length,

dot: dot,

cross: cross

}

})();

console.log(vectorProduct.add([1, 1], [1, 0]))

console.log(vectorProduct.multiply([3.5, -2], 2))

console.log(vectorProduct.length([3, -4]))

console.log(vectorProduct.dot([1, 0], [0, -1]))

console.log(vectorProduct.cross([3, 7], [1, 0]))

|  |
| --- |
| let solution = (() => { |
|  | return { |
|  | add: (firstVector, secondVector) => [firstVector[0] + secondVector[0], firstVector[1] + secondVector[1]], |
|  | multiply: (vector, scalar) => [vector[0] \* scalar, vector[1] \* scalar], |
|  | length: vector => Math.sqrt(vector[0] \* vector[0] + vector[1] \* vector[1]), |
|  | dot: (firstVector, secondVector) => firstVector[0] \* secondVector[0] + firstVector[1] \* secondVector[1], |
|  | cross: (firstVector, secondVector) => firstVector[0] \* secondVector[1] - firstVector[1] \* secondVector[0] |
|  | } |
|  | })(); |
|  |  |
|  | console.log(solution.dot([2, 3], [2, -1])); |

(function () {

function add(vector1, vector2) {

return [vector1[0] + vector2[0], vector1[1] + vector2[1]];

}

function multiply(vector1, scolar) {

return [vector1[0] \* scolar, vector1[1] \* scolar];

}

function length(vector1) {

return Math.sqrt(vector1[0] \* vector1[0] + vector1[1] \* vector1[1]);

}

function dot(vector1, vector2) {

return vector1[0] \* vector2[0] + vector1[1] \* vector2[1];

}

function cross(vector1, vector2) {

return vector1[0] \* vector2[1] - vector1[1] \* vector2[0];

}

return {

add: add,

multiply: multiply,

length: length,

dot: dot,

cross: cross

}

})()

## 6.Breakfast Robot

let breakfastRobot = (

//In Judge must be paste without this above

function makeBreakfast(){

let microelementStock = {

protein: 0,

carbohydrate: 0,

fat: 0,

flavour: 0

};

let recipes = {

apple/\*'apple'\*/: {

carbohydrate: 1,

flavour: 2

},

coke: {

carbohydrate: 10,

flavour: 20

},

burger: {

carbohydrate: 5,

fat: 7,

flavour:3

},

omelet: {

protein: 5,

fat: 1,

flavour: 1

},

cheverme: {

protein: 10,

carbohydrate: 10,

fat: 10,

flavour: 10

}

};

return readCommand;

function readCommand(commandString){

let commandLine = commandString.split(' ');

let command = commandLine[0];

switch(command){

case 'restock':

let microelement = commandLine[1];

let microelementQuantity = Number(commandLine[2]);

return restock(microelement, microelementQuantity);

case 'prepare':

let breakfast = commandLine[1];

let breakfastQuantity = commandLine[2];

return prepare(breakfast, breakfastQuantity);

case 'report':

return report();

}

}

function restock(microelement, quantity){

microelementStock[microelement] += Number(quantity);

return 'Success';

}

function prepare(recipe, quantity){

let breakfast = recipes[recipe];

for(let microelement in breakfast){

if (microelementStock[microelement] < breakfast[microelement] \* Number(quantity)){

return `Error: not enough ${microelement} in stock`;

}

}

Object.keys(breakfast).forEach(microelement => microelementStock[microelement] -= breakfast[microelement] \* Number(quantity));

return 'Success';

}

function report(){

return `protein=${microelementStock.protein} carbohydrate=${microelementStock.carbohydrate} fat=${microelementStock.fat} flavour=${microelementStock.flavour}`;

}

}//In Judge must be paste without this below

)() //This IIFE is for local tests

console.log(breakfastRobot('restock flavour 50'))

console.log(breakfastRobot('prepare coke 4'))

console.log(breakfastRobot('report'))

console.log('Next Test: make another tests in comments!')

// console.log(breakfastRobot('restock carbohydrate 10'))

// console.log(breakfastRobot('restock flavour 10'))

// console.log(breakfastRobot('prepare apple 1'))

// console.log(breakfastRobot('restock fat 10'))

// console.log(breakfastRobot('prepare burger 1'))

// console.log(breakfastRobot('report'))

// console.log('Next Test: make another tests in comments!')

// console.log(breakfastRobot('prepare cheverme 1'))

// console.log(breakfastRobot('restock protein 10'))

// console.log(breakfastRobot('prepare cheverme 1'))

// console.log(breakfastRobot('restock carbohydrate 10'))

// console.log(breakfastRobot('prepare cheverme 1'))

// console.log(breakfastRobot('restock fat 10'))

// console.log(breakfastRobot('prepare cheverme 1'))

// console.log(breakfastRobot('restock flavour 10'))

// console.log(breakfastRobot('prepare cheverme 1'))

// console.log(breakfastRobot('report'))

|  |
| --- |
| let robot = ( |
|  | /\* Judge Submission Start \*/ |
|  | function name() { |
|  | let ingredients = { |
|  | protein: { |
|  | name: 'protein', |
|  | quantity: 0 |
|  | }, |
|  | carbohydrate: { |
|  | name: 'carbohydrate', |
|  | quantity: 0 |
|  | }, |
|  | fat: { |
|  | name: 'fat', |
|  | quantity: 0 |
|  | }, |
|  | flavour: { |
|  | name: 'flavour', |
|  | quantity: 0 |
|  | } |
|  | } |
|  |  |
|  | let mealCooking = { |
|  | apple: (quantity) => useRequiredIngredients([ |
|  | { element: ingredients.carbohydrate, amount: quantity }, |
|  | { element: ingredients.flavour, amount: quantity \* 2 } |
|  | ]), |
|  | coke: (quantity) => useRequiredIngredients([ |
|  | { element: ingredients.carbohydrate, amount: quantity \* 10 }, |
|  | { element: ingredients.flavour, amount: quantity \* 20 } |
|  | ]), |
|  | burger: (quantity) => useRequiredIngredients([ |
|  | { element: ingredients.carbohydrate, amount: quantity \* 5 }, |
|  | { element: ingredients.fat, amount: quantity \* 7 }, |
|  | { element: ingredients.flavour, amount: quantity \* 3 } |
|  | ]), |
|  | omelet: (quantity) => useRequiredIngredients([ |
|  | { element: ingredients.protein, amount: quantity \* 5 }, |
|  | { element: ingredients.fat, amount: quantity }, |
|  | { element: ingredients.flavour, amount: quantity } |
|  | ]), |
|  | cheverme: (quantity) => useRequiredIngredients([ |
|  | { element: ingredients.protein, amount: quantity \* 10 }, |
|  | { element: ingredients.carbohydrate, amount: quantity \* 10 }, |
|  | { element: ingredients.fat, amount: quantity \* 10 }, |
|  | { element: ingredients.flavour, amount: quantity \* 10 } |
|  | ]), |
|  | } |
|  |  |
|  | function useRequiredIngredients(requiredIngredients) { |
|  | for (let i = 0; i < requiredIngredients.length; i++) { |
|  | if (requiredIngredients[i].element.quantity < requiredIngredients[i].amount) { |
|  | returnTakenElements(i); |
|  | return `Error: not enough ${requiredIngredients[i].element.name} in stock`; |
|  | } |
|  |  |
|  | requiredIngredients[i].element.quantity -= requiredIngredients[i].amount; |
|  | } |
|  |  |
|  | return 'Success'; |
|  |  |
|  | function returnTakenElements(indexOfMissingElement) { |
|  | for (let i = indexOfMissingElement - 1; i >= 0; i--) { |
|  | requiredIngredients[i].element.quantity += requiredIngredients[i].amount; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | let commands = { |
|  | restock: (microelement, quantity) => { |
|  | ingredients[microelement].quantity += Number(quantity); |
|  | return 'Success'; |
|  | }, |
|  | prepare: (recipe, quantity) => { |
|  | let meal = mealCooking[recipe.toLowerCase()]; |
|  | if (meal) { |
|  | return meal(Number(quantity)); |
|  | } |
|  |  |
|  | return `Error: recipe for ${recipe} does not exists!`; |
|  | }, |
|  | report: () => Object.keys(ingredients) |
|  | .map(name => `${name}=${ingredients[name].quantity}`) |
|  | .join(' ') |
|  | } |
|  |  |
|  | return function (command) { |
|  | if (command === undefined) { |
|  | return; |
|  | } |
|  |  |
|  | let cmdTokens = command.split(' '); |
|  | let cmd = commands[cmdTokens[0]]; |
|  | if (cmd) { |
|  | return cmd(cmdTokens[1], cmdTokens[2]); |
|  | } |
|  |  |
|  | return 'Error: Command does not exists!'; |
|  | } |
|  | } |
|  | /\* Judge Submission End \*/ |
|  | )() // IIFE - for local tests |
|  |  |
|  | console.log('Test 1'); |
|  | console.log(robot("restock flavour 50")); |
|  | console.log(robot("prepare coke 4")); |
|  | robot() |
|  |  |
|  | console.log('\nTest 2'); |
|  | console.log(robot("restock carbohydrate 10")); |
|  | console.log(robot("restock flavour 10")); |
|  | console.log(robot("prepare apple 1")); |
|  | console.log(robot("restock fat 10")); |
|  | console.log(robot("prepare burger 1")); |
|  | console.log(robot("report")); |

|  |
| --- |
| () => { |
|  | let ingredientStock = { |
|  | protein: 0, |
|  | carbohydrate: 0, |
|  | fat: 0, |
|  | flavour: 0 |
|  | }; |
|  |  |
|  | let recipes = { |
|  | 'apple': { |
|  | carbohydrate: 1, |
|  | flavour: 2 |
|  | }, |
|  | 'coke': { |
|  | carbohydrate: 10, |
|  | flavour: 20 |
|  | }, |
|  | 'burger': { |
|  | carbohydrate: 5, |
|  | fat: 7, |
|  | flavour: 3 |
|  | }, |
|  | 'omelet': { |
|  | protein: 5, |
|  | fat: 1, |
|  | flavour: 1 |
|  | }, |
|  | 'cheverme': { |
|  | protein: 10, |
|  | carbohydrate: 10, |
|  | fat: 10, |
|  | flavour: 10 |
|  | }, |
|  | }; |
|  |  |
|  | function restock(ingredient, quantity) { |
|  | ingredientStock[ingredient] += Number(quantity); |
|  | return 'Success'; |
|  |  |
|  | } |
|  |  |
|  | function prepare(recipe, quantity) { |
|  | let meal = recipes[recipe]; |
|  |  |
|  | for (let key of Object.keys(meal)) { |
|  | if (ingredientStock[key] < meal[key] \* Number(quantity)) { |
|  | return `Error: not enough ${key} in stock`; |
|  |  |
|  | } |
|  |  |
|  | } |
|  | Object.keys(meal).forEach(key => ingredientStock[key] -= meal[key] \* Number(quantity)); |
|  | return 'Success'; |
|  |  |
|  | } |
|  |  |
|  | function report() { |
|  | return `protein=${ingredientStock.protein} carbohydrate=${ingredientStock.carbohydrate} fat=${ingredientStock.fat} flavour=${ingredientStock.flavour}`; |
|  | } |
|  |  |
|  | return function commandParser(str) { |
|  | let cmdInfo = str.split(' '); |
|  | let cmd = cmdInfo.shift(); |
|  | switch (cmd) { |
|  | case "prepare": |
|  | return prepare(cmdInfo[0], Number(cmdInfo[1])); |
|  | case "restock": |
|  | return restock(cmdInfo[0], Number(cmdInfo[1])); |
|  | case "report": |
|  | return report(); |
|  |  |
|  | } |
|  |  |
|  | } |
|  | } |

## 7.Monkey Patcher

function monkeyPatch(commandName){

let balance = this.upvotes - this.downvotes;

let totalVoteCount = this.upvotes + this.downvotes;

let commands = {

upvote: () =>this.upvotes++,

downvote: () => this.downvotes++,

score: () => {

let upvotesToReport = this.upvotes;

let downvotesToReport = this.downvotes;

if(totalVoteCount > 50){

let obfuscatedVotes = Math.ceil(Math.max(upvotesToReport, downvotesToReport) \* 0.25);

upvotesToReport += obfuscatedVotes;

downvotesToReport += obfuscatedVotes;

}

return [upvotesToReport, downvotesToReport, balance, getRating.call(this)];

function getRating() {

if(totalVoteCount < 10){

return 'new';

}

if(this.upvotes > totalVoteCount \* 0.66){

return 'hot';

} else if(balance >= 0 && this.upvotes > 100){

return 'controversial';

} else if(balance < 0){

return 'unpopular';

} else {

return 'new';

}

}

}

}

return commands[commandName]();

}

let post = {

id: '3',

author: 'email',

content: 'wazaaaaa',

upvotes: 100,

downvotes: 100

}

monkeyPatch.call(post, 'upvote')

monkeyPatch.call(post, 'downvote')

console.log(monkeyPatch.call(post, 'score'))

for (let i = 0; i < 50; i++) {

monkeyPatch.call(post, 'downvote')

}

console.log(monkeyPatch.call(post, 'score'))

|  |
| --- |
| function monkeyPatcher(commandName) { |
|  | let balance = this.upvotes - this.downvotes; |
|  | let totalVotes = this.upvotes + this.downvotes; |
|  |  |
|  | let commands = { |
|  | upvote: () => this.upvotes++, |
|  | downvote: () => this.downvotes++, |
|  | score: () => { |
|  | let up = this.upvotes; |
|  | let down = this.downvotes; |
|  |  |
|  | if (totalVotes > 50) { |
|  | let valueToAdd = totalVotes > 50 |
|  | ? Math.ceil(Math.max(up, down) \* 0.25) |
|  | : 0; |
|  |  |
|  | up += valueToAdd; |
|  | down += valueToAdd; |
|  | } |
|  |  |
|  | return [up, down, balance, getRating.call(this)]; |
|  |  |
|  | function getRating() { |
|  | if (totalVotes < 10) { |
|  | return 'new'; |
|  | } |
|  | if (this.upvotes > (this.upvotes + this.downvotes) \* 0.66) { |
|  | return 'hot'; |
|  | } else if (balance >= 0 && (this.upvotes > 100 || this.downvotes > 100)) { |
|  | return 'controversial'; |
|  | } else if (this.upvotes < this.downvotes) { |
|  | return "unpopular"; |
|  | } else { |
|  | return 'new'; |
|  | } |
|  | } |
|  | } |
|  | } |
|  |  |
|  | return commands[commandName](); |
|  | } |
|  |  |
|  | let post = { |
|  | id: '3', |
|  | author: 'emil', |
|  | content: 'wazaaaaa', |
|  | upvotes: 100, |
|  | downvotes: 100 |
|  | }; |
|  |  |
|  | monkeyPatcher.call(post, 'upvote'); |
|  | monkeyPatcher.call(post, 'downvote'); |
|  | console.log(monkeyPatcher.call(post, 'score')); // [127, 127, 0, 'controversial'] |
|  |  |
|  | for (let i = 0; i < 50; i++) { |
|  | monkeyPatcher.call(post, 'downvote'); |
|  | } |
|  |  |
|  | console.log(monkeyPatcher.call(post, 'score')); // [139, 189, -50, 'unpopular'] |

## 8.\* Euclid’s Algorithm

function findGCD(x, y){

let gcd;

if(x === y){

gcd = x;

} else if(x > y){

gcd = findGCD((x - y), y);

} else if(x < y){

gcd = findGCD(x, (y - x));

}

return gcd;

}

console.log(findGCD(252, 105))

|  |
| --- |
| /\* Euclid’s Algorithm - Greatest Common Divisor \*/ |
|  |  |
|  | /\* Reminder Implementation \*/ |
|  | function printGreatestCommonDivisor(firstNumber, secondNumber) { |
|  | let temp = 0; |
|  |  |
|  | while (secondNumber != 0) |
|  | { |
|  | temp = secondNumber; |
|  | secondNumber = firstNumber % secondNumber; |
|  | firstNumber = temp; |
|  | } |
|  |  |
|  | return firstNumber; |
|  | } |
|  |  |
|  | /\* Original - Division Implementation \*/ |
|  | function printGreatestCommonDivisor(firstNumber, secondNumber) { |
|  | while (firstNumber != secondNumber) |
|  | { |
|  | if (firstNumber > secondNumber) { |
|  | firstNumber -= secondNumber; |
|  | } else { |
|  | secondNumber -= firstNumber; |
|  | } |
|  | } |
|  |  |
|  | return firstNumber; |
|  | } |
|  |  |
|  | console.log(printGreatestDivisor(105, 252)); |
|  | console.log(printGreatestDivisor(252, 105)); |

|  |
| --- |
| function euclidGCD(num1, num2) { |
|  | var gcd; |
|  |  |
|  | if (num1 === num2) { |
|  | gcd = num1; |
|  | } else if (num1 > num2) { |
|  | gcd = euclidGCD((num1 - num2), num2); |
|  | } else if (num1 < num2) { |
|  | gcd = euclidGCD(num1, (num2 - num1)); |
|  | } |
|  |  |
|  | return gcd; |
|  | } |
|  |  |
|  | console.log('GCD is: ' + euclidGCD(252, 105)); |

## 9.\*\*\* Kepler’s Problem

function calculateEccentricAnomaly(currentMeanAnomaly, orbitalEccenrticity){

let eccentricalAnomaly = approximate(currentMeanAnomaly, orbitalEccenrticity, 0);

console.log(Number(eccentricalAnomaly.toFixed(9)));

function approximate(meanAnomaly, orbitalEccenrticity, seriesCount){

if(Math.abs(currentMeanAnomaly - (meanAnomaly - orbitalEccenrticity \* Math.sign(meanAnomaly))) < 1e-9 || seriesCount > 200) {

return meanAnomaly;

}

return approximate(meanAnomaly - (meanAnomaly - orbitalEccenrticity \* Math.sin(meanAnomaly) - currentMeanAnomaly) / (1 - orbitalEccenrticity \* Math.cos(meanAnomaly)), orbitalEccenrticity, ++seriesCount);

}

}

calculateEccentricAnomaly(1, 0)

calculateEccentricAnomaly(3.1415926535, 0.75)

calculateEccentricAnomaly(0.25, 0.99)

calculateEccentricAnomaly(4.8, 0.2)

|  |
| --- |
| function keplersProblem(mean, ecc) { |
|  | console.log(Number(approximate(mean, ecc, 0).toFixed(9))); |
|  |  |
|  | function approximate(E0, ecc, series) { |
|  | if (Math.abs(mean - (E0 - ecc \* Math.sign(E0))) < 1e-9 || series > 200) { |
|  | return E0; |
|  | } |
|  |  |
|  | return approximate(E0 - (E0 - ecc \* Math.sin(E0) - mean) / (1 - ecc \* Math.cos(E0)), ecc, ++series); |
|  | } |
|  | } |
|  |  |
|  | keplersProblem(1, 0); |