27)Intro

Objects are used to store related information in a single place.

28)Using Objects with functions

Here we will see how we can pass objects to functions and how we can return object from function. this is how we want to pass values to functions-

let myBook = {

title: '1984',

author: 'George Orwell',

pageCount: 326

};

let otherBook = {

title: 'A peoples History',

author: 'Howard Zinn',

pageCount: 723

};

let getSummary = function (book) {

console.log(`${book.title} by {book.author}`);

};

getSummary(myBook);

getSummary(otherBook);

now lets see how to return object from function. why would we do that. By returning a object , we can return multiple values.

30)Object Refrences

let myAccount = {

name: 'Andrew mead',

expenses: 0,

income: 0

};

let addExpense = function(account,amount) {

account.expenses = account.expenses + amount;

console.log(account);

}

addExpense(myAccount, 2.50);

console.log(myAccount);

code-

**{ name: 'Andrew mead', expenses: 2.5, income: 0 }**

**{ name: 'Andrew mead', expenses: 2.5, income: 0 }**

here we pass a object to a function then we modify it. Now this modification in function actually modifies our original object.

When we pass a object into a function like we do it here, what we get as argument value is’nt just clone of object with exact same properties and values, it’s actually a refrence to exact same object in memory. Here our variable as well as function argument is just a pointer to the object .however if we do this-

let myAccount = {

name: 'Andrew mead',

expenses: 0,

income: 0

};

let addExpense = function(account,amount) {

// account.expenses = account.expenses + amount;

account = {};

console.log(account);

}

addExpense(myAccount, 2.50);

console.log(myAccount);

output –

**{}**

**{ name: 'Andrew mead', expenses: 0, income: 0 }**

Here we made account to point to different location in memory. But global variabale still points to same location. so that binding between myAccount and account is broken. This thing does not only happen when we pass object to function. if do this-

let myAccount = {

name: 'Andrew mead',

expenses: 0,

income: 0

};

let myAccount2 = myAccount;

when we change one object, that change is also reflected in second. this is because they both point to same location in memory. However if ee do this-

let myAccount = {

name: 'Andrew mead',

expenses: 0,

income: 0

};

let myAccount2 = myAccount;

myAccount2 = {};

myAccount2.income = 200000;

now myAccount2 points to diifrent location in memory. Now changes made on it, wnt be reflected in myAccount1.

32)Methods

Method is object property whose value is a function. we get access to objects property inside that function. to do that we use **this** keyword. Inside of method, we have access to this keyword, this refrences the object, on which method is defined on. Code-

let restaurant = {

name: 'ASB',

guestCapacity: 75,

guestCount: 0,

checkAvalaible: function(partySize) {

console.log(this);

}

};

restaurant.checkAvalaible(4);

here we print this, our whole restaurant object is printed on console. Similarly we can use this to access object proeprties. Like , this.name etc.

33)Exploring String Methods

includes

trim

here is link to official documentation-

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String>

here you can see all the methods and properties.

To go to official docs, just google this- **mdn string** and open first link.

34)Exploring Number methods

let num = 103.941;

console.log(num.toFixed(2));

documentation-

google- mdn number and open first link. For numbers we have math object in js. lets see docs. Google it-

math MDN and open first link.

35)Constant Variables

Here we will talk about alternate way of creating variables in js. till now we have been using **let** keyword to create variables. Difference between let based variable and const based variable is you cannot reassign a constant after it has been created. Code-

const isRaining = true;

isRaining = false;

console.log(isRaining);

here we will get error because we are trying to reassign value to const based variable. This code will run , if we use let instead of const. why is this useful.

Many times we create a variable and we know that we are not going to reassign that variable, in these cases it is better to use a const over let.

Now its important to remember that this only applies when we are trying to reassign variable’s value. imagine our variable is a object. Code-

const person = {

age: 27

};

person = {};

console.log(person);

here we will get error because we are reassigning the value to person variable. We are making to point to some other location in memory. But we can still manipulate the object.

Code-

const person = {

age: 27

};

person.age = 26;

console.log(person);

this is valid code.

That is it. That was difference between const and let keywords. That was only difference. Everything else including scoping and valid variable name is identical between 2.

36)Bonus: Variables with var

var was old way of creating variables in js. here we will see that why we should prefer let and const instead of var.

the first ting that we can do with var and we can’t do with const and let is to redeclare a variable, that’s already been created. Ex-

var name = 'sumeet';

var name = 'sood';

console.log(name);

if we try to do it with let or const-

let name = 'sumeet';

let name = 'sood';

console.log(name);

**SyntaxError: Identifier 'name' has already been declared**

This can be a problem. We can accidentally create a variable that already exist.so this is first reason , you might want to avoid var.

Second reason is var is function scoped and not block scoped. That means things like if statements which create code blocks don’t actually create a new scope when working with var. code-

if (10 ===10) {

var name = 'sumeet';

}

console.log(name);

there is no function in program so there is only one scope in program. While let and const are block scoped. Same thing is not possible with let and const. so that was 2 big difference.

The last big difference has to do with accessing a variable before declaring it. Code-

console.log(age);

var age = 10;

**undefined**

while if we use let-

console.log(age);

let age = 10;

we get error-

**ReferenceError: age is not defined**

Reason for this is variable hoisting does not work in case of variables declared with let and const.