ES-2015 Exercise

DESTRUCTURING

1. Use array and object destructuring feature to create variable that hold values as shown below.

```
```

const iPhone11 = { name: 'iPhone 11', manufacturer: 'Apple', price: 699,

    specs: {
        color: 'White',
        memory: {

            value: 128,
            unit: 'MB'

        },

        cameras: {

            front: '12 MP Wide', rear:
            '12 MP Ultra Wide'

        },

        availableColors: ['Black', 'Green', 'Yellow', 'Purple', 'Red', 'White']
    },

    calculateDiscoutedPrice: function(percentage)
        { return this.price * (100 - percentage) / 100;

    }

}

// create the variables using destructuring - the variables should have values as shown below.

// destructure here...

// below line logs - iPhone 11 Apple 128 12 MP Ultra Wide Green
console.log(name, brand, ram, rearCamera, secondColor);
```



```
const iPhone11 = {

   name: 'iPhone 11',
```
```

```
    manufacturer: 'Apple',
    price: 699,
    specs: {
        color: 'White',
        memory: {
            value: 128,
            unit: 'MB'
        },
        cameras: {
            front: '12 MP Wide',
            rear: '12 MP Ultra Wide'
        },
        availableColors: [ 'Black', 'Green', 'Yellow', 'Purple', 'Red', 'White' ]
    },
    calculateDiscoutedPrice: function( percentage ) {
        return this.price * ( 100 - percentage ) / 100;
    }
}

const {
    name ,
    manufacturer: brand,

    specs: {

        memory: {
            value: ram,

        },
        cameras: {
          rear: rearCamera
        },

        availableColors: [ , secondColor, , , , ]
    }
} = iPhone11;
console.log( name, brand, ram, rearCamera, secondColor );
```

2.Write a standalone function ```printPhoneDetails``` that prints a description of the phone with discounted price as in an earlier exercise. The function is passed the iPhone11 object but it destructures to create only the required variables when accepting the object as argument.

```
// iPhone11.calculateDiscoutedPrice = iPhone11.calculateDiscoutedPrice.bind( iPhone11 )
printPhoneDetails.call( iPhone11 ); // Logs "Apple iPhone11 is available at a 10% discounted
rate of $629.1"
```

```
function printPhoneDetails(){
    const {calculateDiscoutedPrice}=this;
    console.log("price is:" +  calculateDiscoutedPrice.call(this, 10));
}
printPhoneDetails.call(iPhone11);
```

3) REST OPERATOR

2. Write a function _*sum*_ that calculates and returns the sum of arguments (assume all arguments are numbers) passed to it, and call it like so. Use a rest parameter to implement this.

~~~
```
var result = sum( 1, 2, 3, 4 );
console.log( result ); // prints 10

var result = sum( 1, 2, 3, 4, 5 ); console.log( result ); // prints 15
~~~

~~~
function sum(...input){


 let sum=0;

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

 {

        sum = sum+input[i];

 }

  return sum;

}

console.log(sum(1,2,3,4,5));
~~~

3. Write function ```max``` that can accept any number of arguments (assume all numbers) and returns the maximum of the numbers. Make use of the rest operator to group the arguments into an array before finding and returning the maximum.

~~~
```

max( 25, 65, 35, 45 ); // 65

max( 25, 65, 35, 75, 45 ); // 75
~~~

~~~
function max(...input1){

        let maximum = input1[0];

        for(let i=1; i<input1.length; i++){

                if(input1[i] > maximum)

                {

                        maximum = input1[i];
~~~

```
        }

    }
    return maximum;

}
console.log(max(1,2,3,4,5));
```

4. Use the spread operator, along with the max function, to find the maximum of values in this array.

```
const numbers = [ 25, 65, 35, 75, 45 ];

function max(...input1){
    let maximum = input1[0];
    for(let i=1; i<input1.length; i++){
        if(input1[i] > maximum)
        {
            maximum = input1[i];
        }

    }
    return maximum;

}
console.log(max(25, 65, 35, 75, 45));
```

5. Use object spread to make a shallow copy of the following object.

* Again, use array and object spread (as required) to create a deep copy of
  the object.
* Test out if making a change to name and front camera details on the
iPhone11 object affects the shallow copy.
* What about the deep copy? Is it affected?

```
const iPhone11 = {
    name: 'iPhone 11',
    manufacturer: 'Apple',
    price: 699,
    specs: {
        color: 'White',
        memory: {
            value: 128,
            unit: 'MB'
        },
        cameras: {
            front: '12 MP Wide',
            rear: '12 MP Ultra Wide'
        },
        availableColors: [ 'Black', 'Green', 'Yellow', 'Purple', 'Red', 'White' ]
    }
};


const iphoneCopy = {
    ...iPhone11 // name: 'John', age: 32, address : john.address, emailids:
  john.emailids
    //age: 33,
    //spouse: 'Jane'
};

iphoneCopy.price= 700;
console.log(iphoneCopy);
console.log(iPhone11);

iPhone11.name = "xyz";
iPhone11.specs.cameras.front = "bhbh";
console.log(iphoneCopy);

const DeepiPhone11 = {

    ...iPhone11,
    specs: {

        ...iPhone11.specs,
        memory: {
            ...iPhone11.specs.memory
        },
        cameras: {
            ...iPhone11.specs.cameras
        },
        availableColors: [ ...iPhone11.specs.availableColors ]
    }
};


console.log(DeepiPhone11);

iPhone11.specs.memory.value = 10;
console.log(iPhone11);
```