Serie N°1: JS Fundamentals

[JavaScript]

Exercise 1: Type coercion in an operand context (in operators such as +, -, *, /, &&, ||).
For each instruction, define:

- The type of each operand
- The possible types of each operand after type conversion
- Possible operation types
- The output

Define the result of each instruction.

N°	*Instructions*	*Possible operations*	*Possible types after conversion*		*Output*
			Operand 1	*Operand 2*	
1	**"10" + 5;**				
2	**2 + "10" + 5;**				
3	**3 + 2 + "10" + 5;**				
6	**"10" + 5 * 2;**				
7	**"5" * 2;**				
8	**true + true;**				
9	**true + "1";**				
10	**true * "1":**				
11	**null + 1:**				
12	**null + "1";**				
13	**null* "1";**				
14	**null * true:**				
15	**null * "true";**				
16	**5 - "2";**				
17	**5 - "abc";**				
18	**5 - false;**				
19	**5 - "false";**				
20	**5 / "true";**				

Exercise 2: Objects

```
let student = {
  firstName: "Oussama",
  lastName: "MESSAOUDI",
  id: "124097240293",
  class: "2023",
  contact: {
    email:"example@gmail.com",
    phoneNumber: "055555555",
    address:"example address"
  },
  fullName: function() {
      return this.firstName+" "+this.lastName;
  }
};
```

Given the student object structure, respond to the questions bellow:

1. Print first name on the console
2. Print contact property
3. Use the fullName property to print the student full name
4. Print email, phone number, and address separately
5. Update student first name and last name
6. Update student email
7. Delete contact property
8. Add back contact property with email

Exercise 3: Using the student object defined in exercise 2, answer the following questions. Note that any field in the object can be null.

1. If ***firstName*** is not null/undefined, print it, else print "value is null"
2. Print student ***email*** if possible, else print null.
3. Print a message in the form of "hello my name is ***firstName***, reach me at ***phoneNumber***".
4. Add a function ***incrementBonus*** that increments ***bonus*** property in student object with a value passed as a parameter.
5. Call ***incrementBonus*** and then print ***bonus***
6. Add the pair key-value ***age-24*** to student object
7. Update ***class*** to 2022
8. Delete ***contact*** from student object
9. Reinsert ***email*** in ***contact*** and add it to student object

```
let student = {
  //old student fields
  incrementBonus: Function(value) {
        this.bonus = (this.bonus ?? 0) + value;
    }
};
console.log(student?.firstName ?? "Value is null");
console.log(student?.contact?.email);
console.log(`hello my name is ${student?.firstName}, reach me at
${student?.contact?.phoneNumber}`);
student.incrementBonus(5);
console.log(student?.bonus);
student.age = 24;
student.class = 2022;
delete student.contact;
student.contact = {
  email: "new_email@gmail.com",
};
```

Exercise 6:

Write three JavaScript functions that detects changes in two ***values*** as follows:

1. ***isValueChanged***: takes two parameters and return:
    a. true if one of the two values is falsy and the other is truthy
    b. true if the two values have different types
    c. if simple types, and have different values return true, else return. False
    d. if arrays: call ***isArrayChanged*** to compare both arrays
    e. if map: call ***isMapChanged***
2. ***isArrayChanged***: takes two arrays and return:
    a. false if both arrays have the same reference
    b. true if arrays have different lengths
    c. check both array value by value, return true if you detect a difference, else return false (use isValueChanged to compare)
3. ***isMapChanged***: takes two objects and return:
    a. false if both arrays have the same reference
    b. true if arrays have different number of keys
    c. check both objects key by key, return true you detect a difference in key values, else return false (use isValueChanged to compare)

*The checking process starts by calling **isValueChanged** to compare two **values***

```js
/**
 * Compare to values `before` and `after`, and return true they are diffrent
 * @param {{}} before
 * @param {{}} after
 * @returns {bool}
 */
function isValueChanged(before, after) {
    if (!before === !!after) return true;//on of two objects is falsy
    if (typeof before != typeof after) return true;
    if (typeof before !== 'object' || typeof after !== 'object') {
        return before !== after;
    }
    if (Array.isArray(before)){
        isArrayChanged(before,after);
    }
    return isMapChanged(before,after);
}
```

```js
/**
 * return true if arrays `before` and `after` are diffrent, else false
 * @param {[]} bafore
 * @param {[]} after
 * @returns {bool}
 */
function isArrayChanged(bafore, after) {
    if (bafore === after) return false;
    if (bafore.length !== after.length) return true;
    for (var i = 0; i < bafore.length; ++i) {
        if (bafore[i] !== after[i]) return true;
    }
    return false;
}
```

```
/**
 * Compare two maps `before` and `after`, return true if all key values are equal, else false
 * @param {{}} before
 * @param {{}} after
 * @returns {bool}
 */
function isMapChanged(before, after) {
    if (before === after) return false;
    if (Object.keys(before).length != Object.keys(after).length) return true;
    for (const key in after) {
        if (!Object.hasOwnProperty.call(before, key)) {
            return true;
        }
        if (isValueChanged(before[key], after[key])) {
            return true;
        }
    }
    return false;
}
```