JavaScript Exercise

1. We have an object storing salaries of our team

```
let salaries = {
    John: 100,
    Ann: 160,
    Pete: 130
}
```

Write the code to sum all salaries and store in the variable sum. Should be 390 in the example above.

```
let sum = 0;
for( let key in salaries){
    sum += salaries[key];
}
console.log(sum);
```

2. Create a function multiplyNumeric(obj) that multiplies all numeric properties of obj by 2

```
// before the call
let menu = {
    width: 200,
    height: 300,
    title: "My menu"
};

multiplyNumeric(menu);

// after the call
menu = {
    width: 400,
    height: 600,
    title: "My menu"
};
```

Please note that multiplyNumeric does not need to return anything. It should modify the object in-place

```
function multiplyNumeric(obj){
    for (let key in menu){
        if (typeof obj[key] == 'number'){
            obj[key] *= 2
        }
    }
}
```

3. Write a function checkEmailId(str) that returns true if str contains '@' and '.', otherwise false. Make sure '@' must come before '.' and there must be some characters between '@' and '.'
The function must be case-insensitive:

```
function checkEmail(str){
    let regexEmail = /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/;
    if (str.match(regexEmail)){
        return true;
    }else{
        return false;
    }
}
```

4. Create a function truncate(str, maxlength) that checks the length of the str and, if it exceeds maxlength – replaces the end of str with the ellipsis character "…", to make its length equal to maxlength.

   The result of the function should be the truncated (if needed) string.

```
truncate("What I'd like to tell on this topic is:", 20) = "What I'd like to te…"

truncate("Hi everyone!", 20) = "Hi everyone!"
```

```
function truncate(str, maxLength){
    return (str.length > maxLength) ? str.slice(0, maxLength - 1) + '...': str;
}
```

5. Let’s try 5 array operations.

   Create an array styles with items “James” and “Brennie”.
   Append “Robert” to the end.
   Replace the value in the middle by “Calvin”. Your code for finding the middle value should work for any arrays with odd length.
   Remove the first value of the array and show it.
   Prepend Rose and Regal to the array.

```
James, Brennie
James, Brennie, Robert
James, Calvin, Robert
Calvin, Robert
Rose, Regal, Calvin, Robert
```

```
      var nameList = ["James", "Brennie"]
nameList.push("Robert");
nameList.splice(Math.floor((nameList.length - 1) / 2), Math.floor((nameList.length -
1) / 2), "Calvin")
nameList.splice(0, 1)
nameList.splice(0, 0, "Rose", "Regal")
```

6. We are transitioning from people swiping their credit card to using a chip. As part of the transition, some credit card companies have moved all of their customers to the new chip card, some are in the process, and some still have yet to comply. We need to implement a system that verifies the card swipe and determine if we should authorize the transaction using a card swipe or if the customer needs to use the chip feature of their card.

   We need you to implement the validateCards() function. We will pass in two lists of strings. We are doing batch authorization so the first parameter is a list of the swiped card number. The second is a list of card prefixes that cannot be processed if the card is swiped.

The function should return a JSON array of credit card objects that contains: Card (String): the card number isValid(boolean): a modified Luhn check (described below) isAllowed (boolean): a check to verify that the number doesn't start with any of the card prefixes.

The modified Luhn check verifies that the card was swiped correctly. We do this by calculating the check digit and comparing it to the last number on the card. We calculate the check digit by taking all but the last digit the card number, take the int value of each individual character, double it, add them together, then divide by 10 and take the reminder. That value is the calculated check digit. If this matches the check digit (last number in the string), then the card number is valid.

Below is an example Let's say we have a card number: 6724843711060148. The last number is 8. We then have the string 672484371106014. We now take each character and convert it to a number: |6|7|2|4|8|4|3|7|1|1|0|6|0|1|4|

Then we double each number: |12|14|4|8|16|8|6|14|2|2|0|12|0|2|8|

Then we sum them up and get 108. We divide by 10 and the remainder is 8. The calculated check digit from our card matches the last number, so we set isValid to true. If it were any other number, we would set it to false.

For the full example, we would have the input parameter of: cardsToValidate (List of one credit card number): 6724843711060148 And bannedPrefixes(string): 11, 3434, 67453, 9 The result would be a JSON array with one credit card object{ [{card:"6724843711060148", isValid:true, isAllowed:true}]

```
function validateCardNumber(number) {
    var regex = new RegExp("^[0-9]{16}$");
    if (!regex.test(number)){
        return false;
    }
    var json = {};
    var dic = {};
    var li = [];
    dic[card] = number.toString();
    dic[isValid] = is_valid(number);
    dic[isAllowed] = is_allowed(number);
    li.push(dic);
    json.push(li);
    return json;
}

function is_valid(val) {
    var sum = 0;
    for (var i = 0; i < val.length; i++) {
        var intVal = parseInt(val.substr(i, 1));
        if (i % 2 == 0) {
            intVal *= 2;
            if (intVal > 9) {
                intVal = 1 + (intVal % 10);
            }
        }
        sum += intVal;
    }
    return (sum % 10) == 0;
}

function is_allowed(val){
    var prefix = [3, 5, 4, 6];
    var num = Math.floor((val / Math.pow(10, 15)) % 10);
    return prefix.includes(num);
}
```

7. Validating Email Addresses with RegEx We consider an email address in the form user@domain.extension to be valid if its domain and extension are hackerrank.com and the value of user satisfies the following constraints:
   a. It starts with between 1 and 6 lowercase English letters denoted by the character class [a-z].
   b. The lowercase letter(s) are followed by an optional underscore, i.e. zero or one occurrence of the character.
   c. The optional underscore is followed by 0 and 4 optional digits denoted by the character class[0-9].

Complete the code in the editor below by replacing the bank ("_____") with a regular expression that matches valid email addresses according to the criteria above. Locked code in the editor prints True for each correct match and False for each incorrect match.

An example of a valid emails is abcdef_1234@hackerrank.com. It has as many of each class of character as possible. The address a1_1@baddomain.com fails for two reasons. First, digits cannot precede the underscore. Second, the domain fails because it is not hackerrank.

Constraints

- 1 <= query <= 10 power of 3.
- 1 <= string length <= 10 power of 3

Input Format Locked stub code reads strings from stdin and processes them with your regex.

The first line contains an integers, query, describing the total number of target strings to match against the regular expression. Each line i of the query subsequent lines contains a string describing the ith email address to validate.

Sample Case 0
Sample Input 0
5
julia@hackerrank.com

julia_@hackerrank.com
julia_0@hackerrank.com
julia0_@hackerrank.com
julia@gmail.com

Sample Output 0

True
True
True
False
False

Explanation 0 We perform the following query = 5 validations:

- julia@hackerrank.com starts with between 1 and 6 lowercase letters and contains zero of the optional characters, so it's valid.
- julia_@hackerrank.com starts with between 1 and 6 lowercase letters, is followed by a single underscore, and contains none of the optional digits, so it's valid.
- julia_0@hackerrank.com starts with between 1 and 6 lowercase letters, is followed by a single underscore, and is followed by between 0 and 4 digits, so its valid.
- julia0_@hackerrank.com has valid lowercase letters followed by a valid digit, but the digit must not precede the underscore.
- julia@gmail.com has a valid username, but its domain and extension do not match hackerrank.com.

```
function validateEmail(email) {
    const re = /^(([^<>()[\]\\.,;:\s@"]+(\.[^<>()[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/;
    return re.test(String(email).toLowerCase());
}
```