Excellent !

Please note:

- Read the questions carefully. Every word is important.
- Feel free to add comments in your answers and read the documentation as you see fit.
- Don't forget to mention the language you are using.
- Do test your code!
- Between parenthesis is the weight of each question, the maximum score is 20

Test 1: Any language (3/20)
Write a program that outputs sequentially the integers from 1 to 99, but on some conditions prints a string instead:

- when the integer is a multiple of 3 print “Open” instead of the number,
- when it is a multiple of 7 print “Source” instead of the number,
- when it is a multiple of both 3 and 7 print “OpenSource” instead of the number.

Language: Javascript
just using a brute force method here

number: number to check if it's a multiple
base: number to check multiplication of

```
function checkMultiple(number, base) {
   return number % base === 0;
}

let num;
const MAX_NUMBER = 100;

for (num = 1; num < MAX_NUMBER; num++) {
   if (checkMultiple(num, 3) && checkMultiple(num, 7)) {
      console.log('OpenSource');
   } else if (checkMultiple(num, 3)){
      console.log('Open');
   } else if (checkMultiple(num, 7)){
      console.log('Source');
   } else {
      console.log(`${num}`);
   }
}
```

[juc] 3

Test 2: Any language (3/20)
Solve the game "Guess a number", find a secret integer between 1 and 1000000
in less than 50 guesses. Write a function that solves the game **without user input** and returns the
solution by using the function verify() which is defined with the following
specification:
function verify(guess: integer) -> integer
Argument:
    guess (integer) the number to verify
Returns:
    0 if the guess is the solution, your program won
    -1 if the solution is smaller than the guess parameter
    1 if the solution is bigger than the guess parameter

Warning: You are not allowed to call verify() more that 50 times or you lose.

Language: Javascript
Created a function guess that returns a function solve that is called to guess the number
how the function guesses the number? by generating a number in a specific range
how the range is specified? it's a simple equation it depends on the return value from verify
if the return value is 0 then it's solved
if it's 1 then we know that the max number in the range should be less than the guessed number
if it's -1 then we know that the min number in the range should be bigger than the guessed number

```
// A function to generate a number in a specified range (using math)
const generateRandomNumber = (min, max) => Math.floor(Math.random() * (max - min) + min);
const guess = (hiddenNumber) => {
   let tries = 1;
   const MAX_NUMBER_OF_TRIES = 50;
   const MIN_RANGE = 1;
   const MAX_RANGE = 1000000;
   function verify(guessedNumber) {
      return guessedNumber === hiddenNumber
         ? 0
         : guessedNumber > hiddenNumber
            ? 1
            : -1;
   }
   return {
      solve(min=MIN_RANGE, max=MAX_RANGE) {
         if (tries >= MAX_NUMBER_OF_TRIES) {
            console.log(`Failed to guess your number I'm such a loser`);
            return;
         }
         let guessedNumber = generateRandomNumber(min, max);
         const verified = verify(guessedNumber);
         if (verified === 0) {
            console.log(`It's ${guessedNumber} got the number  in ${tries} tries Nailed it!!`);
            return;
         }
         if (verified === 1) max = guessedNumber - 1;
         else min = guessedNumber + 1;
         tries += 1;
         this.solve(min, max);
      }
   }
}

// to use it
const numberToGuess = 148623;
guess(numberToGuess).solve();

// if you ever decided to challenge the program with a different range you can pass it to solve
guess(50).solve(min = 10, max = 100);
```

[juc] 0, "-1 if the solution is smaller than the guess parameter" your verify implementation returns 1. Also as you use random to guess a number in your boundaries, there is a narrow (mal)chance you pick close numbers 50 times in a row, use a bisection instead to garantee log2(n).

Test 3: Any language (3/20)
Write a function that takes a list of strings and returns the sum of the list items that represent an integer (skipping the other items)

Language: Javascript
this is iterative using reduce

```
const sumOfIntegers = (list) => {
   return list.reduce((sum, listItem) => {
      // checking first if a number using isNan (Nan for not a number)
      // isNan returns false for numbers even if it was of type string!!
      // if true then just return the sum as it is
      if (isNaN(listItem)) return sum;
      // if it's a number add it to sum and return the result
      // using '+' before the item to convert to integer (it also trims trailing spaces)
      return sum += (+listItem);
   }, 0)
}

console.log(sumOfIntegers(['53', 'hjj', '4j4hg7', '  47  ']));   // 53 + 47  = 100
```

[juc] 3

Test 4: Any language (1/20)
Write a recursive version of the previous function (or an iterative version if you have already done a recursive version).

Language: Javascript
this is recursive

```
const sumOfIntegers = (list) => {
   // the stop condition is empty list
   if (list.length === 0) return 0;
   // using splice to truncate the first item [0] in the list
   return isNaN(list[0])? sumOfIntegers(list.splice(1))
                        : (+list[0]) + sumOfIntegers(list.splice(1));
}
```

[juc] 1

Test 5: SQL (3/20)

- Write pseudo-SQL statements to create database tables to store the products of a basic webshop. Each product has a name, a price, a creation date and may belong to several categories. Categories have a name and a flag to indicate whether the category is private or public.
- Write a SQL query to find the list of products that belong to more than 5 public categories.

couldn't find a standard syntax for pseudo-Sql so I made this up :D

```
 table products (
    id number auto increment primay key,
    name string,
    price number,
    created_at date default now,
)

//suppose if flag is true then it's private
table categories (
    id number auto increment primay key,
    name string,
    flag boolean default false
)

// there is a many to many relationship between products and categories so a junction table is needed

table products_categories (
    product_id number forign key products(id),
    category_id number forign key categories(id)
)

// left join could be also used here too since we already know that the ids in the junction table (products_categories)
 exists in products/categories
But maybe using inner join is faster and safe!!

SQL Query

SELECT p.*
FROM products_categories pc
INNER JOIN products p ON p.id = pc.product_id
INNER JOIN categories c ON c.id = pc.category_id
WHERE c.flag = false
GROUP BY p.id
HAVING COUNT(p.id) > 5
```

[juc] 3, excellent ! "pseudo-sql" just mean "you can use whatever syntax you want as long as it looks like sql"

Test 6: Any language (1/20)
Write a program to download the contents of https://www.sap.com/belgique/index.html (the SAP homepage for Belgium), and then save the contents of the page to a new local file, with all occurrences of "SAP" replaced by "Odoo".

Language: Python

```
from urllib.request import urlopen
import re

url = 'https://www.sap.com/belgique/index.html';
regex = r'sap'
subst = 'Odoo'

with  open(r'manipulated-file.htm', 'w') as manipulatedFile, urlopen(url) as con:
    # read the file line by line, do the "manipulation" then write in the file
    for line in con.readlines():
        # the line comes as a binary so we need to decode it first
        # the encoding method the link uses is utf8
        decodedLine = line.decode('utf8')
        # using Regex to replace sap by Odoo
        manipulatedLine = re.sub(regex, subst, decodedLine, flags=re.IGNORECASE)
        manipulatedFile.write(manipulatedLine)

print('Everything is done!');
```

[juc] 1

Test 7: Any language (2/20)
You have a huge file named "data.bin" that does not fit in memory; code a program that deletes every 7th byte of it. truncate can be used to change its size.

Language: Python

```
# if data.bin is huge that memory can't afford it
# then maybe it's better to create a temp file to hold the data until truncating is done
# instead of storing the data in a var or something (memory)
# then after that remove data.bin file and rename the temp file to data.bin

import os

with open("data.bin", "rb") as original, open("temp.bin", 'wb') as temp:
    sixBytes = original.read(6)

    while sixBytes:
        temp.write(sixBytes)
        print('7th: ', original.read(1))
        sixBytes = original.read(6)

os.remove('data.bin')
os.rename('temp.bin', 'data.bin')
```

[juc] 2, excellent !

Test 8: Regular Expression (2/20)
Write a regular expression to match strings containing both "Odoo" and "#rules" in any order.

 using lookahead assertion magic

```
 /(?=.*Odoo)(?=.*#rules).*/g
```

 Lookahead here doesn't really match anything it just returns a result (match or no match)
 So here it checks if there're "zero or many letters followed by Odoo" and "zero or many letters followed by #rules"
 then that .* at the end used to match the whole string if checks are true (of course)

[juc] 2

Test 9: Javascript (1/20)
Write a function that, when called, returns an array for which each element is a letter of the alphabet, from "A" to "Z" (exactly once, in order and upper case). Your code cannot contain the character ' (quote), " (double quote) or ` (back quote)

 using ASCII codes
 A - Z ascii codes is 65 - 90
 then use String.fromCharCode() to convert it to a string

```
const charsArray = () => {
    const chars = [];
    for (let charCode = 65; charCode <= 90; charCode++) {
        chars.push(String.fromCharCode(charCode));
    }
    return chars;
}

console.log(charsArray());
```

[juc] 1

Test 10: Unix (1/20)
Write a C program that roughly measures the overhead of a mode switch between the kernel and userspace on a UNIX/Linux system (without using the pthread API). Please indicate the assumptions your rough estimation is based on.

```
// used syscall() as a benchmark to make the system switch to kernel mode
// and used time.h to calculate the time to do so

#include <time.h>
#include <stdio.h>
#include <sys/syscall.h>

int main () {
   clock_t start, end, total;

   start = clock();

//  SYS_write specifies the system call number which differs on each OS
// '1' represents number of fields to Write
// '15' represents the size in bytes
   syscall(SYS_write, 1, "test test test\n", 15);

   end = clock();

   total = (long double)(end - start) / CLOCKS_PER_SEC;
   // the number is insanely small so using e may help
   printf("Elapsed Time: %Le\n", total);

   return(0);
}
```

[juc] 1