





# JavaScript Loop Exercises

## Level 1

1. Viết hàm `printNumbers(a, b)` nhận hai giá trị đầu vào `a, b` là hai số nguyên dương bất kỳ ( $a < b$ ), in ra các số trong khoảng từ `a` - `b` theo thứ tự tăng dần
2. Viết hàm `printNumbers(a, b)` nhận hai giá trị đầu vào `a, b` là hai số nguyên dương bất kỳ ( $a < b$ ), in ra các số trong khoảng từ `a` - `b` ~~theo thứ tự giảm dần~~
3. Viết hàm `sumOfOddNumbers(a, b)` nhận hai giá trị đầu vào `a, b` là hai số nguyên dương bất kỳ, tính và **trả về kết quả** là tổng của các số lẻ trong khoảng từ `a` đến `b` (bao gồm cả `a` và `b`).  
 Lưu ý trường hợp `a` có thể nhỏ hơn `b`
4. Viết hàm `divisor(number)` nhận một giá trị đầu vào `number` là một số bất kỳ, in ra các ước số của `number`
5. Viết hàm `factorial(number)` nhận một giá trị đầu vào `number` là một số nguyên dương bất kỳ, tính và **trả về kết quả** là giai thừa của `number`
6. Viết hàm `countFolding(thickness)` nhận một giá trị đầu vào `thickness` là độ dày (đơn vị mét), tính số lần gấp một tờ giấy có độ dày 0.1mm để đạt được độ dày lớn hơn hoặc bằng với `thickness` truyền vào
7. Viết hàm `countYears(dad, son)` nhận hai giá trị đầu vào:
  - `dad` là số tuổi của cha (cha lớn hơn con ít nhất 18 tuổi :v)
  - `son` là số tuổi của conTính xem sau bao nhiêu năm nữa thì tuổi cha gấp 2 lần tuổi con
8. Bài toán dân gian gà và chó, tìm ra số chân gà và chó:  
Vừa gà vừa chó  
Bó lại cho tròn  
Ba mươi sáu con  
Một trăm chân chẵn

## Level 2

1. Viết hàm `isPrime(number)` nhận một giá trị đầu vào và `number` là một số nguyên dương bất kỳ, kiểm tra và **trả về kết quả** số đó có phải là số nguyên tố (`true`) hay không (`false`)  
 Số nguyên tố là số chỉ chia hết cho 1 và chính nó, không chia hết cho bất kỳ số nào khác
2. Viết hàm `sumOfPrimes(a, b)` nhận hai giá trị đầu vào `a, b` là hai số nguyên dương bất kỳ
3. Viết hàm `fibonacci(n)` nhận một giá trị đầu vào `n` là một số nguyên dương bất kỳ, in ra dãy `n` số Fibonacci  
 Fibonacci là dãy số bắt đầu bằng 0 và 1, các số tiếp theo bằng tổng 2 số đứng trước nó
4. Viết hàm `isPalindrome(n)` nhận một giá trị đầu vào `n` là một số nguyên dương bất kỳ, kiểm tra và **trả về kết quả** cho biết số đó có phải số Palindrome (`true`) hay không (`false`)  
 Palindrome là các số (hay chuỗi) giống nhau khi đảo ngược thứ tự các con số, ví dụ: 121, 303, 4004
5. Viết hàm `sumOfPalindrome(n)` nhận một giá trị đầu vào `n` là một số nguyên dương bất kỳ, tính và **trả về kết quả** là tổng của các số Palindrome trong khoảng từ 1 -> `n`
6. Viết hàm `reverseNumber(n)` nhận một giá trị đầu vào `n` là một số nguyên bất kỳ, đảo ngược các chữ số và **trả về kết quả** (không sử dụng chuỗi)