# Bonus: Generator Functions and Iterators

Problems for in-class lab for the [“JavaScript Advanced” course @ SoftUni](https://softuni.bg/courses/javascript-advanced). Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/345/>.

## Reverse Iterator

Write JS function that takes an array as parameter and returns an Iterator object. The Iterator must iterate over the items of the array in reverse order.

### Input / Output

Your function needs to take an array as a parameter. As output, **return** an object as described in the Iterator JavaScript Protocol.

### Examples

|  |  |
| --- | --- |
| Sample Input | Output |
| let iterator = reverseArrayIterator([10, 20, 30]);  while (true) {  let item = iterator.next();  if (item.done) break;  console.log(item.value);  } | 30  20  10 |

## Iterable

Write a JS function that takes an array as parameter and returns an Iterable object. The object must iterate over the items of the array in reverse order.

### Input / Output

Your function needs to take an array as a parameter. As output, **return** an object as described in the Iterable JavaScript Protocol.

### Examples

|  |  |
| --- | --- |
| Sample Input | Output |
| for (let item of reverseArrayIterable([10, 20, 30])) {  console.log(x);  } | 30  20  10 |

## Reverse Generator

Write a JS **Generator** function that takes an array as parameter and **yields** the elements of the array in reverse order.

### Input / Output

Your function needs to take an array as a parameter. As output, **yield** an element of the array.

### Examples

|  |  |
| --- | --- |
| Sample Input | Output |
| let arr = [10, 20, 30];  for (let item of reverseArrayGenerator(arr)) {  console.log(item);  } | 30  20  10 |

## Iterate HTML Tags

Write a JS **Generator** function that takes a string as parameter and **yields** all HTML tags from that string.

### Input / Output

Your function needs to take a string as a parameter. As output, **yield** a containing an HTML tag.

### Examples

|  |  |
| --- | --- |
| Sample Input | Output |
| let html = `<html><body>  <p align='center'><span lang='en'>Hello</span>, HTML  </p> Bye, bye</body></html>`;  for (let tag of extractTags(html)) {  console.log(tag);  } | <html>  <body>  <p align='center'>  <span lang='en'>  </span>  </p>  </body>  </html> |

## Object Iterator

Create a function which takes in an object and returns an iterator for it. The iterator should have a method next() which returns an object containing properties value - holding the next element and done (true or false) - signifying if the iterator has finished. The iterator should enumerate all keys of the object **alphabetically in descending order**.

### Examples

|  |  |
| --- | --- |
| Sample Input | Output |
| **let *obj*** = {**age**: 27, **name**: **"pesho"**, **book**: **"Lord of the Rings"**};  **let *iterator*** = *makeIterable*(***obj***);  **while**(**true**){  **let *res*** = ***iterator***.next();  **if**(***res***.**done**) **break**;  **console**.log(***res***.**value**);  } | name  book  age |
| **let *obj*** = {**name**: **"gosho"**, **"13"**: **true**, **book**: **"Lord of the Drinks"**, 2: 2, **age**: 15, **passportNumber**: 12345678};  **let *iterator*** = *makeIterable*(***obj***);  **while**(**true**){  **let *res*** = ***iterator***.next();  **if**(***res***.**done**) **break**;  **console**.log(***res***.**value**);  } | passportNumber  name  book  age  2  13 |

### Template

Submit in the judge the JS code (implementation) of the makeIterable function. Here's an example template:

|  |
| --- |
| make-iterable.js |
| **function** *makeIterable*(object) {  *//****TODO***  **return** {  next: **function** () {  *//****TODO***  }  }  } |

### Hints

* The comparing should be done alphabetically so you can use localeCompare. Keep in mind that numbers also should be compared by alphabetically.

## Fibonacci Generator

Write a JS Generator function that yields numbers from the Fibonacci sequence. Assume the first two elements in the sequence are **1** and **1**.

### Input / Output

There will be no input. As output, your function needs to **yield** the values as described.

### Examples

|  |  |
| --- | --- |
| Sample Input | Output |
| let fib = fibonacci();  console.log(fib.next().value);  console.log(fib.next().value);  console.log(fib.next().value);  console.log(fib.next().value);  console.log(fib.next().value);  console.log(fib.next().value);  console.log(fib.next().value); | 1  1  2  3  5  8  13 |
| let fib = fibonacci();  for (let number of fib) {  console.log(number);  } | 1  1  2  3  5  … *// series is infinite* |

## Random Generator

Write a JS Generator function that yields pseudo-random numbers between 0 and 99 by supplied seed. When using the same seed to initialize two different generators, they should produce an identical sequence. Use the following formula to calculate the value of the next number:



Where ***xn*** is the next number in the sequence, ***xn-1*** is the previous number in the sequence, ***Mod*** is the modulo (remainder) operator and ***p*** = 4871 and ***q*** = 7919. When calculating the first element in the sequence, use the seed for the value of ***xn-1***. See the examples for detailed calculations.

### Input / Output

As input, your function will receive a seed value as a number. As output you need to **yield** the values as described.

### Examples

|  |  |  |
| --- | --- | --- |
| Sample Input | Output | Explanation |
| let rnd = random(100);  for (let i = 0; i < 10; i++) {  console.log(rnd.next().value);  } | 0  2  29  89  34  76  47  83  30  40 | When calculating the first number, we take the value of the seed for ***xn-1*** in the formula. For ***xn*** we get:  (100 \* 100) % (4871 \* 7919) = 10000  Since we need a number between 0 and 99, we further apply modulo 100 to this result and yield it:  10000 % 100 = 0  The next number starts with 10000 as the value of ***xn-1*** to arrive at:  100002 % 38573449 = 22853102 % 100 = 2 |

## \*\*Look and Say Generator

Have you heard of the Look and Say Sequence? It is an interesting sequence that works like this:  
**1** -> **11** -> **21** -> **1211** -> **111221** -> **312211**

You start off with a number in this instance **1** - it is the first element in the sequence - the next element **11** is calculated in the following way - you take the previous element (**1**) and you **compress all** **consecutive same digits** **into one** and in front of it you write the **number of digits compressed** the **blue** number here is the count of the digits compressed and the **red** number is the digit that was compressed. This can be more easily explained with the 5th and 6th elements - **111221** -> we have **3** consecutive **1**'s so we compress them into **31** (**3** times the digit **1**) then we have **2** consecutive **2**'s, we compress them into **22** (**2** times the digit **2**) and finally we have **1** times the digit **1** - we compress it into **11** (**1** time the digit **1**) thus we get **312211**.

Create a generator function which takes as a parameter a number and can produce the next elements from the sequence as a **STRING**.

### Examples

|  |  |
| --- | --- |
| Sample Input | Output |
| **let *lookSequence*** = *lookAndSay*(1);  **console**.log(***lookSequence***.next().**value**);  **console**.log(***lookSequence***.next().**value**);  **console**.log(***lookSequence***.next().**value**);  **console**.log(***lookSequence***.next().**value**);  **console**.log(***lookSequence***.next().**value**); | 11  21  1211  111221  312211 |
| **let *lookSequence*** = *lookAndSay*(113);  **console**.log(***lookSequence***.next().**value**);  **console**.log(***lookSequence***.next().**value**);  **console**.log(***lookSequence***.next().**value**);  **console**.log(***lookSequence***.next().**value**);  **console**.log(***lookSequence***.next().**value**); | 2113  122113  11222113  21322113  121113222113 |

### Template

Submit in the judge the JS code (implementation) of the lookAndSay function. Here's an example template:

|  |
| --- |
| make-iterable.js |
| **function** \* *lookAndSay*(start) {  *//****TODO***  **yield** nextElement;  } |

### Hints

* You can check more about the [Look and Say Sequence](https://en.wikipedia.org/wiki/Look-and-say_sequence) in Wikipedia.
* Think of a way to get all consecutive digits from a number, treating it as a string may be easier.