Does JavaScript have a method like “range()” to generate a range within the supplied bounds?

Asked 8 years, 10 months ago Active 27 days ago Viewed 508k times

673

129

In PHP, you can do...

```
range(1, 3); // Array(1, 2, 3)
range("A", "C"); // Array("A", "B", "C")
```

That is, there is a function that lets you get a range of numbers or characters by passing the upper and lower bounds.

Is there anything built-in to JavaScript natively for this? If not, how would I implement it?

javascript arrays

edited Mar 10 '18 at 21:37
Evan Carroll
38.6k 25 156 266

asked Oct 9 '10 at 2:37
alex
355k 177 794 924

1 Prototype.js has the `$R` function, but other than that I don't really think so. – Yi Jiang Oct 9 '10 at 2:42

This (related) question has some excellent answers: stackoverflow.com/questions/6299500/… – btk Feb 25 '15 at 14:47

`Array.from("ABC") //['A', 'B', 'C']` This is the closest thing I can find for the second part of the question. – Andrew_1510 May 16 '16 at 1:51

@Andrew_1510 You could use `split("")` there also – alex May 22 '16 at 17:46

When lover bound is zero this oneliner: `Array.apply(null, { length: 10 }).map(eval.call, Number)` – csharpfolk Jul 25 '16 at 16:32

52 Answers

1107

Numbers

```
[...Array(5).keys()];
 => [0, 1, 2, 3, 4]
```

Character iteration

```
String.fromCharCode(...[...Array('D'.charCodeAt(0) - 'A'.charCodeAt(0) +
1).keys()].map(i => i + 'A'.charCodeAt(0)));
 => "ABCD"
```

Iteration

```
for (const x of Array(5).keys()) {
  console.log(x, String.fromCharCode('A'.charCodeAt(0) + x));
}
 => 0,"A" 1,"B" 2,"C" 3,"D" 4,"E"
```

As functions

```
function range(size, startAt = 0) {
    return [...Array(size).keys()].map(i => i + startAt);
}

function characterRange(startChar, endChar) {
    return String.fromCharCode(...range(endChar.charCodeAt(0) -
            startChar.charCodeAt(0), startChar.charCodeAt(0)))
}
```

As typed functions

```
function range(size:number, startAt:number = 0):ReadonlyArray<number> {
    return [...Array(size).keys()].map(i => i + startAt);
}
```

lodash.js `_.range()` function

```
_.range(10);
 => [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
_.range(1, 11);
 => [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
_.range(0, 30, 5);
 => [0, 5, 10, 15, 20, 25]
_.range(0, -10, -1);
 => [0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
String.fromCharCode(..._.range('A'.charCodeAt(0), 'D'.charCodeAt(0) + 1));
 => "ABCD"
```

Old non es6 browsers without a library:

```
Array.apply(null, Array(5)).map(function (_, i) {return i;});
 => [0, 1, 2, 3, 4]
```

```
console.log([...Array(5).keys()]);
```

Run code snippet Expand snippet

Thanks.

(ES6 credit to nils petersohn and other commenters)

edited Apr 12 at 17:03

answered Apr 7 '12 at 1:05
Ben
11.4k 2 13 16

65 Because if it's useful anywhere it is probably useful in JS. (JS can do functional programming type stuff, which can benefit from a range(0 statement. That and a thousand other reasons it might be useful in some semirare case) – Lodewijk May 18 '13 at 19:47

12 `Array(5).fill()` is also mappable – nils petersohn Mar 21 '17 at 12:54

275

For numbers you can use ES6 `Array.from()` , [which works in everything these days]() except IE:

Shorter version:

```
Array.from({length: 20}, (x,i) => i);
```

Longer version:

```
Array.from(new Array(20), (x,i) => i)
```

which creates an array from 0 to 19 inclusive. This can be further shortened to one of these forms:

```
Array.from(Array(20).keys())
// or
[...Array(20).keys()]
```

Lower and upper bounds can be specified too, for example:

```
Array.from(new Array(20), (x,i) => i + *lowerBound*)
```

An article describing this in more detail: http://www.2ality.com/2014/05/es6-array-methods.html

edited Jan 9 '18 at 2:50
dman
4,940 12 65 138

answered Apr 10 '15 at 10:47
CapK
4,411 2 18 24

49 The first example can even be simplified to [...Array(20).keys()] – Delapouite Nov 20 '15 at 17:39

23 Slightly more succinct than the `Array.from()` method, and faster than both: `Array(20).fill().map((  i) => i)` – Stu Cox Feb 20 '16 at 8:34

Join Stack Overflow to learn, share knowledge, and build your career.

Sign up with email Sign up with Google Sign up with Facebook ✕

1 @icc97 Yes, linters may complain, although in JavaScript omitting a function argument defined to be the same as passing `undefined`, so `fill()` (with no argument) isn't *wrong* per se. The fill value isn't used in that solution, so if you like you could use `fill(0)` to save a few characters. – Stu Cox Jan 16 at 8:17

100

My new favorite form (*ES2015*)

```
Array(10).fill(1).map((x, y) => x + y)
```

And if you need a function with a `step` param:

```
const range = (start, stop, step = 1) =>
  Array(Math.ceil((stop - start) / step)).fill(start).map((x, y) => x + y * step)
```

edited Jan 6 at 18:06

answered Jul 6 '17 at 19:12
Kutyel
5,054 2 24 51

5 let range = (start, stop, step=1) => Array(stop - start).fill(start).map((x, y) => x + y * step) – rodfersou Feb 25 '18 at 2:32

4 @rodfersou FYI: your example is wrong. `stop` is not actually the stop / end position but count / distance. (no offense, just to make people aware of the typo) – F Lekschas Jul 26 '18 at 2:45

4 For the confused - due to rodfersou's edit after F Lekschas' comment, his code is now correct. – eedrah Sep 13 '18 at 17:15

1 The argument you pass into `Array(Math.ceil((stop - start) / step) + 1)`, needs the `+1` at the end, to really mimic php's "inclusive" behaviour. – Johan Dettmar Oct 26 '18 at 8:16

3 This is the top answer that actually answers the full question of a Javascript function that fully implements a `range` method. All the others currently above this (except for lodash's `_.range`) implement basic iterators rather than an actual range function with start, stop and step – icc97 Jan 6 at 17:53

Here's my 2 cents:

edited Mar 18 at 7:24
Kutyel
5,054 2 24 51

answered Oct 21 '13 at 22:51
jflood.net
1,930 1 15 19

1 Excellent use of high order functions. – Farzad YZ May 18 '16 at 10:19

4 This is actually wrong because the question is asking for start & end values. Not start & count/distance. – James Robey Mar 31 '18 at 6:52

67

✓

It works for characters and numbers, going forwards or backwards with an optional step.

```
var range = function(start, end, step) {
    var range = [];
    var typeofStart = typeof start;
    var typeofEnd = typeof end;

    if (step === 0) {
        throw TypeError("Step cannot be zero.");
    }

    if (typeofStart == "undefined" || typeofEnd == "undefined") {
        throw TypeError("Must pass start and end arguments.");
    } else if (typeofStart != typeofEnd) {
        throw TypeError("Start and end arguments must be of same type.");
    }

    typeof step == "undefined" && (step = 1);

    if (end < start) {
        step = -step;
    }

    if (typeofStart == "number") {

        while (step > 0 ? end >= start : end <= start) {
            range.push(start);
            start += step;
        }
```

```
            start = start.charCodeAt(0);
            end = end.charCodeAt(0);

            while (step > 0 ? end >= start : end <= start) {
                range.push(String.fromCharCode(start));
                start += step;
            }

        } else {
            throw TypeError("Only string and number types are supported");
        }

        return range;

    }
```

jsFiddle.

If augmenting native types is your thing, then assign it to `Array.range` .

Show code snippet

edited May 1 '17 at 15:00
mplungjan
94.8k 23 134 192

answered Oct 9 '10 at 2:54
alex
355k 177 794 924

43

Simple range function:

```
function range(start, stop, step) {
    var a = [start], b = start;
    while (b < stop) {
        a.push(b += step || 1);
    }
    return a;
}
```

Join Stack Overflow to learn, share knowledge, and build your career.

Sign up with email   Sign up with Google   Sign up with Facebook

3 PLUS UNO for usable and readable. Best code snippet I've seen in a long time. – monsto Dec 7 '15 at 17:26

1 This doesn't work when `step != 1`, the `while` condition needs to take `step` into account. My updated version with a default `step` value: function range(start, stop, step){ step = step || 1 var a=[start], b=start; while((b+step) < stop){ console.log("b: " + b + ". a: " + a + "."); b+=step; a.push(b); } return a; } – daveharris Oct 25 '17 at 21:51

@daveharris I added a default step above, `(step || 1)`. – Mr. Polywhirl May 9 '18 at 14:57

34

```
Array.range= function(a, b, step){
    var A= [];
    if(typeof a== 'number'){
        A[0]= a;
        step= step || 1;
        while(a+step<= b){
            A[A.length]= a+= step;
        }
    }
    else{
        var s= 'abcdefghijklmnopqrstuvwxyz';
        if(a=== a.toUpperCase()){
            b=b.toUpperCase();
            s= s.toUpperCase();
        }
        s= s.substring(s.indexOf(a), s.indexOf(b)+ 1);
        A= s.split('');
    }
    return A;
}


    Array.range(0,10);
    // [0,1,2,3,4,5,6,7,8,9,10]

    Array.range(-100,100,20);
    // [-100,-80,-60,-40,-20,0,20,40,60,80,100]

    Array.range('A','F');
    // ['A','B','C','D','E','F')
```

82.6k 25 89 122

34

OK, in JavaScript we don't have a `range()` function like **PHP**, so we need to create the function which is quite easy thing, I write couple of one-line functions for you and separate them for **Numbers** and **Alphabets** as below:

for **Numbers**:

```
function numberRange (start, end) {
  return new Array(end - start).fill().map((d, i) => i + start);
}
```

and call it like:

```
numberRange(5, 10); //[5, 6, 7, 8, 9]
```

for **Alphabets**:

```
function alphabetRange (start, end) {
  return new Array(end.charCodeAt(0) - start.charCodeAt(0)).fill().map((d, i) =>
String.fromCharCode(i + start.charCodeAt(0)));
}
```

and call it like:

```
alphabetRange('c', 'h'); //["c", "d", "e", "f", "g"]
```

edited Sep 19 '18 at 10:40

answered Jul 27 '17 at 15:46

Alireza
60.1k 14 197 129

2 I think there are off-by-one errors in these functions. Should be `Array(end - start + 1)` , and `Array(end.charCodeAt(0) - start.charCodeAt(0) +`

23

```
function range(start, end, step, offset) {

  var len = (Math.abs(end - start) + ((offset || 0) * 2)) / (step || 1) + 1;
  var direction = start < end ? 1 : -1;
  var startingPoint = start - (direction * (offset || 0));
  var stepSize = direction * (step || 1);

  return Array(len).fill(0).map(function(_, index) {
    return startingPoint + (stepSize * index);
  });

}

console.log('range(1, 5)=> ' + range(1, 5));
console.log('range(5, 1)=> ' + range(5, 1));
console.log('range(5, 5)=> ' + range(5, 5));
console.log('range(-5, 5)=> ' + range(-5, 5));
console.log('range(-10, 5, 5)=> ' + range(-10, 5, 5));
console.log('range(1, 5, 1, 2)=> ' + range(1, 5, 1, 2));
```

Run code snippet Expand snippet

here is how to use it

range (Start, End, Step=1, Offset=0);

- inclusive - forward `range(5,10)   // [5, 6, 7, 8, 9, 10]`
- inclusive - backward `range(10,5)   // [10, 9, 8, 7, 6, 5]`
- step - backward `range(10,2,2)   // [10, 8, 6, 4, 2]`
- exclusive - forward `range(5,10,0,-1)   // [6, 7, 8, 9]  not 5,10 themselves`
- offset - expand `range(5,10,0,1)   // [4, 5, 6, 7, 8, 9, 10, 11]`
- offset - shrink `range(5,10,0,-2)   // [7, 8]`
- step - expand `range(10,0,2,2)   // [12, 10, 8, 6, 4, 2, 0, -2]`

Basically I'm first calculating the length of the resulting array and create a zero filled array to that length, then fill it with the needed values

- `(step || 1)` => And others like this means use the value of `step` and if it was not provided use `1` instead
- We start by calculating the length of the result array using `(Math.abs(end - start) + ((offset || 0) * 2)) / (step || 1) + 1)` to put it simpler (difference* offset in both direction/step)
- After getting the length, then we create an empty array with initialized values using `new Array(length).fill(0);` check here
- Now we have an array `[0,0,0,..]` to the length we want. We map over it and return a new array with the values we need by using `Array.map(function() {})`
- `var direction = start < end ? 1 : 0;` Obviously if `start` is not smaller than the `end` we need to move backward. I mean going from 0 to 5 or vice versa
- On every iteration, `startingPoint + stepSize * index` will gives us the value we need

edited May 23 '17 at 11:47
Community ♦
1 1

answered Aug 13 '15 at 13:20
azerafati
11.5k 3 43 57

8 Handy, most certainly. Simple? I beg to differ; regardless that you make it a one liner. Coming from Python this is a shock. – PascalVKooten Mar 2 '16 at 16:02

@PascalvKooten, yeah of course it would have been great if there was built-in method for that like python I guess, but this was the simplest one I could come by. And it has proven to be handy in my projects. – azerafati Mar 3 '16 at 7:02

1 I ended up using it anyway... – PascalVKooten Mar 3 '16 at 12:56

Posting a painfully complex code snippet like that, especially as a one-liner and with no explanation of how it works? Poor example of a *good* SO answer, regardless of whether or not it "works". – Madbreaks Dec 21 '16 at 19:34

1 @Madbreaks, yea you're right. I've been naive to make it a one liner. just wanted to give everyone a quick and easy solution – azerafati Dec 22 '16 at 7:16

```
var range = (l,r) => new Array(r - l).fill().map((_,k) => k + l);
```

What's the `_` ? – nikk wong Apr 13 '16 at 18:41

@nikkwong, the `_` is just a name of argument in the mapping callback. You know, in some languages you would use the `_` as a name to point out that the variable is not used. – Artur Klesun Apr 13 '16 at 19:34

Here though, `_` isn't passed through the arguments to `range` . Why not? – nikk wong Apr 13 '16 at 20:55

2 Very neat! Although, it's important to note it doesn't work on any IE or Opera. – Rafael Xavier May 22 '16 at 0:45

4 This answer needs explanation, as it stands its a poor fit for SO. – Madbreaks Dec 21 '16 at 19:57

17

Using Harmony spread operator and arrow functions:

```
var range = (start, end) => [...Array(end - start + 1)].map((_, i) => start + i);
```

Example:

```
range(10, 15);
[ 10, 11, 12, 13, 14, 15 ]
```

answered Dec 30 '14 at 9:21
c.P.u1
12k 3 36 37

that's the best answer! – Henry H. Dec 20 '15 at 9:25

1 Not the fastest though. – mjwrazor May 9 '17 at 21:05

What does underscore '_' symbol represents in this case? – Oleh Berehovskyi Feb 27 at 9:24

@OlehBerehovskyi It means a lambda function parameter that you have no intent of actually using. A linter that warns about unused variables should ignore it. – Micah Zoltu Mar 16 at 6:23

If you'd like to double-check, the definitive resource is the ECMA-262 Standard.

edited Jul 19 '18 at 5:38

answered Oct 9 '10 at 2:53
Mike Dinescu
40.9k 8 87 128

While I'm sure a perfectly good answer in 2010, this should no longer be considered the best approach. You should not extend built in types, like Prototype.js tended to do 👍 – Dana Woodman Jul 19 '18 at 3:34

@DanaWoodman thanks for bringing this up - I've updated the answer to take out the reference to Prototype.js since that is indeed pretty much obsolete in 2018 – Mike Dinescu Jul 19 '18 at 5:39

3 Well this didn't help at all. – Pithikos Apr 1 at 19:09

@Pithikos I see this question has been edited since it was originally asked and the OP wanted to know if there is a native range function in JS. – Mike Dinescu Apr 3 at 2:57

13

Did some research on some various Range Functions. **Checkout the jsperf comparison** of the different ways to do these functions. Certainly not a perfect or exhaustive list, but should help :)

The Winner is...

```
function range(lowEnd,highEnd){
    var arr = [],
    c = highEnd - lowEnd + 1;
    while ( c-- ) {
        arr[c] = highEnd--
    }
    return arr;
}
range(0,31);
```

Technically its not the fastest on firefox, but crazy speed difference (imho) on chrome makes up for it.

Also interesting observation is how much faster chrome is with these array functions than firefox. **Chrome is at least 4 or 5 times**

Note that this was compared against range functions that included a step size parameter – binaryfunt Aug 24 '17 at 13:01

12

You can use lodash or Undescore.js `range` :

```
var range = require('lodash/range')
range(10)
// -> [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ]
```

Alternatively, if you only need a consecutive range of integers you can do something like:

```
Array.apply(undefined, { length: 10 }).map(Number.call, Number)
// -> [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ]
```

In ES6 `range` can be implemented with generators:

```
function* range(start=0, end=null, step=1) {
  if (end == null) {
    end = start;
    start = 0;
  }

  for (let i=start; i < end; i+=step) {
    yield i;
  }
}
```

This implementation saves memory when iterating large sequences, because it doesn't have to materialize all values into an array:

```
for (let i of range(1, oneZillion)) {
  console.log(i);
```

39.6k 20 89 87

The ES6 part is now the correct answer to this question. I would recommend removing the other parts, which are covered by other answers. – joews Nov 8 '15 at 15:24

generators are somewhat strange if used outside a loop though: x=range(1, 10);//{} x;//{}// looks like an empty map WTF!?! x.next().value;// OK 1 ;x[3] // undefined, only with real array – Anona112 Dec 8 '16 at 20:06

@Anona112 you can use `Array.from` to convert generators to array instances and inspect the output. – Paolo Moretti Dec 8 '16 at 21:43

10

An interesting challenge would be to write the *shortest* function to do this. Recursion to the rescue!

```
function r(a,b){return a>b?[]:[a].concat(r(++a,b))}
```

Tends to be slow on large ranges, but luckily quantum computers are just around the corner.

An added bonus is that it's obfuscatory. Because we all know how important it is to hide our code from prying eyes.

To truly and utterly obfuscate the function, do this:

```
function r(a,b){return (a<b?[a,b].concat(r(++a,--b)):a>b?[]:[a]).sort(function(a,b)
{return a-b})}
```

answered Aug 6 '14 at 17:24
Rick Hitchcock
30.3k 2 31 59

4 Short != simple, but simpler is better. Here's an easier to read version: `const range = (a, b) => (a>=b) ? [] : [a, ...range(a+1, b)]` , using ES6 syntax – nafg Dec 26 '16 at 7:27

1 @nafg: `const range = (a, b, Δ = 1) => (a > b) ? [] : [a, ...range(a + Δ, b, Δ)];` . Also upvoting the whole answer for the comment. – 7vujy0f0hy Dec 29 '17 at 16:38

Join Stack Overflow to learn, share knowledge, and build your career.

Sign up with email | Sign up with Google | Sign up with Facebook

9

```
const RANGE = (a,b) => Array.from((function*(x,y){
  while (x <= y) yield x++;
})(a,b));

console.log(RANGE(3,7));  // [ 3, 4, 5, 6, 7 ]
```

Or, if we only need iterable, then:

```
const RANGE_ITER = (a,b) => (function*(x,y){
  while (x++< y) yield x;
})(a,b);

for (let n of RANGE_ITER(3,7)){
  console.log(n);
}
```

answered May 28 '17 at 17:46
Hero Qu
379 5 5

I would code something like this:

8

```
function range(start, end) {
    return Array(end-start).join(0).split(0).map(function(val, id) {return id+start});
}

range(-4,2);
// [-4,-3,-2,-1,0,1]

range(3,9);
// [3,4,5,6,7,8]
```

It behaves similarly to Python range:

Sasha Zezulinsky
967 12 13

8

A rather minimalistic implementation that heavily employs ES6 can be created as follows, drawing particular attention to the `Array.from()` static method:

```
const getRange = (start, stop) => Array.from(
  new Array((stop - start) + 1),
  (_, i) => i + start
);
```

answered Feb 15 '17 at 3:03
IsenrichO
1,584 2 11 26

As a side note, I've created a Gist in which I made an "enhanced" `getRange()` function of sorts. In particular, I aimed to capture edge cases that might be unaddressed in the bare-bones variant above. Additionally, I added support for alphanumeric ranges. In other words, calling it with two supplied inputs like `'C'` and `'K'` (in that order) returns an array whose values are the sequential set of characters from the letter 'C' (inclusive) through the letter 'K' (exclusive): `getRange('C', 'K'); // => ["C", "D", "E", "F", "G", "H", "I", "J"]` – IsenrichO Mar 14 '17 at 3:10

you don't need the `new` keyword – Soldeplata Saketos Mar 14 at 17:41

7

Though this is not from *PHP*, but an imitation of `range` from ***Python***.

```
function range(start, end) {
    var total = [];

    if (!end) {
        end = start;
        start = 0;
    }
```

```
console.log(range(10)); // [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
console.log(range(0, 10)); // [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
console.log(range(5, 10)); // [5, 6, 7, 8, 9]
```

answered Jul 9 '14 at 13:06
user642922

+1 for the fastest. with an array of -36768 - 36768, took 3ms, 2nd place was 13 ms and has IDE red lines. – mjwrazor May 9 '17 at 21:12

7

`range(start,end,step)` : With ES6 Iterators

You only ask for an upper and lower bounds. **Here we create one with a step too.**

You can easily create `range()` generator function which can function as an iterator. This means you don't have to pre-generate the entire array.

```
function * range ( start, end, step = 1 ) {
  let state = start;
  while ( state < end ) {
    yield state;
    state += step;
  }
  return;
};
```

Now you may want to create something that pre-generates the array from the iterator and returns a list. This is useful for functions that accept an array. For this we can use `Array.from()`

```
const generate_array = (start,end,step) =>
  Array.from( range(start,end,step) );
```

But when something desires an iterator (or gives you the option to use an iterator) you can easily create one too.

```
for ( const i of range(1, Number.MAX_SAFE_INTEGER, 7) ) {
  console.log(i)
}
```

Special Notes

- If you use Ramda, they have their own `R.range` as does Lodash

edited Mar 1 at 4:28

answered Jan 28 at 6:00
Evan Carroll
38.6k 25 156 266

6

As far as generating a numeric array for a given range, I use this:

```
function range(start, stop)
{
    var array = [];

    var length = stop - start;

    for (var i = 0; i <= length; i++) {
        array[i] = start;
        start++;
    }

    return array;
}

console.log(range(1, 7));  // [1,2,3,4,5,6,7]
console.log(range(5, 10)); // [5,6,7,8,9,10]
console.log(range(-2, 3)); // [-2,-1,0,1,2,3]
```

Setting `array = []` inside the loop may not give you what you want. – alex May 23 '15 at 2:36

@alex, thank you. You're right, I also forgot to increment the "start" parameter on each pass of the loop. It's fixed now. – jhaskell May 24 '15 at 2:56

It still won't produce the desired output, if I want range 5-10, it will give me `[5, 6, 7, 8, 9, 10, 11, 12, 13, 14]`, I would expect only the first half of that array. – alex May 24 '15 at 7:55

@alex, thank you again, I had not considered a length constraint based on input. See updated version. – jhaskell May 25 '15 at 16:46

6

Using Harmony generators, supported by all browsers except IE11:

```
var take = function (amount, generator) {
    var a = [];

    try {
        while (amount) {
            a.push(generator.next());
            amount -= 1;
        }
    } catch (e) {}

    return a;
};

var takeAll = function (gen) {
    var a = [],
        x;

    try {
        do {
            x = a.push(gen.next());
        } while (x);
    } catch (e) {}

    return a;
};

var range = (function (d) {
    if (typeof d.step === "undefined") {
        if (unlimited) {
            d.step = 1;
        }
    } else {
        if (typeof d.from !== "string") {
            if (d.from < d.to) {
                d.step = 1;
            } else {
                d.step = -1;
            }
        } else {
            if (d.from.charCodeAt(0) < d.to.charCodeAt(0)) {
                d.step = 1;
            } else {
                d.step = -1;
            }
        }
    }

    if (typeof d.from === "string") {
        for (let i = d.from.charCodeAt(0); (d.step > 0) ? (unlimited ? true : i <=
d.to.charCodeAt(0)) : (i >= d.to.charCodeAt(0)); i += d.step) {
            yield String.fromCharCode(i);
        }
    } else {
        for (let i = d.from; (d.step > 0) ? (unlimited ? true : i <= d.to) : (i >=
d.to); i += d.step) {
            yield i;
        }
    }
});
```

Examples

take

Example 1.

`take` only takes as much as it can get

```
[100, 105, 110, 115, 120]
```

Example 2.

`to` not neccesary

```
take(10, range( {from: 100, step: 5} ) )
```

returns

```
[100, 105, 110, 115, 120, 125, 130, 135, 140, 145]
```

takeAll

Example 3.

`from` not neccesary

```
takeAll( range( {to: 5} ) )
```

returns

```
[0, 1, 2, 3, 4, 5]
```

Example 4.

```
takeAll( range( {to: 500, step: 100} ) )
```

returns

```
[0, 100, 200, 300, 400, 500]
```

Example 5.

```
takeAll( range( {from: 'z', to: 'a'} ) )
```

returns

1 Nice, but it could benefit from being a bit more readable. – alex Oct 30 '12 at 21:47

@alex: suggestions welcome :) hehehe – Janus Troelsen Oct 31 '12 at 0:20

Edited with my suggestions :) – Xotic750 Jun 12 '13 at 12:50

+1 for the approach. To @alex's point, not having ternary operations (especially not nested) in the `for` clause would improve readability here. – Justin Johnson May 27 '14 at 6:03

5

you can use `lodash` function `_.range(10)` https://lodash.com/docs#range

edited Feb 2 '15 at 18:07
Bhargav Rao ♦
32.4k 21 94 115

answered Feb 2 '15 at 15:47
Ikan
144 1 5

5

d3 also has a built-in range function. See https://github.com/mbostock/d3/wiki/Arrays#d3_range:

> **d3.range([start,]stop[, step])**
>
> Generates an array containing an arithmetic progression, similar to the Python built-in range. This method is often used to iterate over a sequence of numeric or integer values, such as the indexes into an array. Unlike the Python version, the arguments are not required to be integers, though the results are more predictable if they are due to floating point precision. If step is omitted, it defaults to 1.

Example:

```
d3.range(10)
// returns [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

answered Nov 30 '15 at 4:59

Join Stack Overflow to learn, share knowledge, and build your career.

Sign up with email Sign up with Google Sign up with Facebook

Thank you so much. I use D3 and was looking for a native JS method, not knowing that I D3 offers it already. – cezar Jul 1 at 7:24

5

... more range, using a generator function.

```
function range(s, e, str){
  // create generator that handles numbers & strings.
  function *gen(s, e, str){
    while(s <= e){
      yield (!str) ? s : str[s]
      s++
    }
  }
  if (typeof s === 'string' && !str)
    str = 'abcdefghijklmnopqrstuvwxyz'
  const from = (!str) ? s : str.indexOf(s)
  const to = (!str) ? e : str.indexOf(e)
  // use the generator and return.
  return [...gen(from, to, str)]
}

// usage ...
console.log(range('l', 'w'))
//=> [ 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w' ]

console.log(range(7, 12))
//=> [ 7, 8, 9, 10, 11, 12 ]

// first 'o' to first 't' of passed in string.
console.log(range('o', 't', "ssshhhooooouuut!!!!"))
// => [ 'o', 'o', 'o', 'o', 'o', 'u', 'u', 'u', 't' ]

// only lowercase args allowed here, but ...
console.log(range('m', 'v').map(v=>v.toUpperCase()))
//=> [ 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V' ]

// => and decreasing range ...
console.log(range('m', 'v').map(v=>v.toUpperCase()).reverse())
```

```
// ... etc, etc.
```

Hope this is useful.

answered Oct 10 '17 at 21:36
John Swindin
69 1 1

4

Complete ES6 implementation using range([start,]stop[, step]) signature:

```
function range(start, stop, step=1){
  if(!stop){stop=start;start=0;}
  return Array.from(new Array(int((stop-start)/step)), (x,i) => start+ i*step)
}
```

If you want automatic negative stepping, add

```
if(stop<start)step=-Math.abs(step)
```

Or more minimalistically:

```
range=(b, e, step=1)=>{
  if(!e){e=b;b=0}
  return Array.from(new Array(int((e-b)/step)), (_,i) => b<e? b+i*step : b-i*step)
}
```

If you have huge ranges look at Paolo Moretti's generator approach

edited Jan 25 '17 at 8:49

answered Oct 8 '16 at 9:32
Anona112
2,223 3 14 28

Join Stack Overflow to learn, share knowledge, and build your career.

Sign up with email Sign up with Google Sign up with Facebook

4

There's an npm module bereich for that ("bereich" is the German word for "range"). It makes use of modern JavaScript's iterators, so you can use it in various ways, such as:

```
console.log(...bereich(1, 10));
// => 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

const numbers = Array.from(bereich(1, 10));
// => [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]

for (const number of bereich(1, 10)) {
  // ...
}
```

It also supports descending ranges (by simply exchanging `min` and `max`), and it also supports steps other than `1` .

Disclaimer: I am the author of this module, so please take my answer with a grain of salt.

edited Apr 1 '18 at 10:46

answered Feb 13 '18 at 3:51

Golo Roden
61.7k 63 217 326

2 Why the downvote? – Golo Roden Apr 5 '18 at 9:42

3

I was surprised to come across this thread and see nothing like my solution (maybe I missed an answer), so here it is. I use a simple range function in ES6 syntax :

```
// [begin, end[
const range = (b, e) => Array.apply(null, Array(e - b)).map((_, i) => {return i+b;});
```

But it works only when counting forward (ie. begin < end), so we can modify it slightly when needed like so :

Join Stack Overflow to learn, share knowledge, and build your career.

Sign up with email Sign up with Google Sign up with Facebook ✕

nha
11.5k 7 63 85

Use `[...Array(e-b)]` while you are at it. – andlrc Feb 9 '16 at 21:22

3

Here's a nice short way to do it in ES6 with numbers only (don't know its speed compares):

```
Array.prototype.map.call(' '.repeat(1 + upper - lower), (v, i) => i + lower)
```

For a range of single characters, you can slightly modify it:

```
Array.prototype.map.call(' '.repeat(1 + upper.codePointAt() - lower.codePointAt()), (v,
i) => String.fromCodePoint(i + lower.codePointAt()));
```

answered Oct 13 '16 at 16:02
Michael Plautz
1,368 3 15 30

3

You can also do the following:

```
const range = Array.from(Array(size)).map((el, idx) => idx+1).slice(begin, end);
```

edited Jul 23 '18 at 4:22
Grant Miller
8,402 13 46 73

answered Jul 23 '18 at 1:12
gty3310
31 2

1 2 next

Join Stack Overflow to learn, share knowledge, and build your career.

Sign up with email Sign up with Google Sign up with Facebook

Would you like to answer one of these unanswered questions instead?