

Lists and Maps

DG Script has two collection types:

- A **list** is an ordered collection, written `[1, 2, 3]`. It is the equivalent of a JavaScript array.
- A **map** is a collection of named values, written `{"name": "Pump", "speed": 3}`. It is the equivalent of a JavaScript object.

Topics on this page:

- [Lists: creating and accessing, list members, sorting, forEach, where, map](#)
- [Maps](#)
- [Looping with for...in](#)
- [Array\(\)](#)

See also: [JSON and XML](#), [Tables](#).

Lists

Creating and Accessing Lists

Create a list with square brackets. Read and change items by position; the first position is 0.

```
row = [1, "two", 3];
print(row[1], "\n");
row[0] = 10;
print(row, "\n");
print(typeof row, "\n");
return row.length;
```

Print output:

```
two
[10, two, 3]
list
```

Output: 3

Things to know:

- Reading a position at or after the end of the list causes an error. Negative positions are not supported.
- Assigning to a position at or after the end causes an error. Use `push` to add items.
- You can shorten or lengthen a list by setting `length`. New positions are filled with `null`.
- Lists cannot have named properties. `list.name = "x"` does not work.

List Members

Member	Description
<code>list.length</code>	The number of items. You can also assign to it
<code>list.first</code> , <code>list.last</code>	The first or last item (properties, not methods). An empty list causes an error
<code>list.push(item)</code>	Adds one item to the end. Same as <code>add</code> . Returns <code>null</code> (not the new length)
<code>list.pushAll(otherList)</code>	Adds all items of <code>otherList</code> to the end
<code>list.pop()</code>	Removes and returns the last item. An empty list causes an error
<code>list.unshift(item)</code>	Inserts one item at the start
<code>list.shift()</code>	Removes and returns the first item. An empty list causes an error
<code>list.indexOf(item)</code>	The position of the first item equal to <code>item</code> , or <code>-1</code> . No type conversion: <code>[1].indexOf("1")</code> is <code>-1</code>
<code>list.join(separator)</code>	A string of all items joined with <code>separator</code> . Without a separator, items are joined with nothing , not with commas as in JavaScript
<code>list.concat(list1, list2, ...)</code>	A new list with the items of this list followed by the items of each argument. Arguments that are not lists are ignored
<code>list.slice(start, end)</code>	The items from <code>start</code> up to (not including) <code>end</code> . <code>end</code> is optional. Negative values are not supported. See the note below
<code>list.splice(start, count, item1, item2, ...)</code>	Removes <code>count</code> items starting at <code>start</code> , inserts the new items at that position, and returns a list of the removed items. <code>count</code> is required
<code>list.sort(...)</code>	Sorts the list in place and returns it. See Sorting
<code>list.forEach(function)</code>	See forEach , where , map
<code>list.where(function)</code>	See forEach , where , map
<code>list.map(function)</code>	See forEach , where , map
<code>list.encodeBase64()</code>	Encodes a list of byte values as Base64. See Encoding
<code>list.query(name)</code> , <code>list.queryAll(name)</code>	Search a list of XML nodes. See JSON and XML

`slice` does not return a real list. It returns a read-only range that prints as `(2, 3)` and does not support list members such as `length`. To get a list, pass it to `concat`: `[] .concat(list.slice(1, 3))`.

```

row = [1, 2];
row.push(3);
row.pushAll([4, 5]);
print(row, "\n");
last = row.pop();
first = row.shift();
row.unshift(0);
print(first, " ", last, " ", row, "\n");
print(row.indexOf(3), " ", row.first, " ", row.last, "\n");
print(row.join(), " ", row.join("-"), "\n");
print([].concat(row.slice(1, 3)), "\n");
removed = row.splice(1, 2, "a", "b", "c");
print(removed, " ", row, "\n");
row.length = 2;
return row.concat([7], [8, 9]);

```

Print output:

```

[1, 2, 3, 4, 5]
1 5 [0, 2, 3, 4]
2 0 4
0234 0-2-3-4
[2, 3]
[2, 3] [0, a, b, c, 4]

```

Output: [0, a, 7, 8, 9]

Sorting

`list.sort()` sorts the list in place and also returns it. There are two ways to call it.

With a compare function, `list.sort(function (a, b) { ... })`: the function returns a negative number if a comes first, a positive number if b comes first, or 0 if they are equal.

With flags, `list.sort(asString, descending, caseInsensitive)` (all optional, default false):

- `asString` — false: sort as numbers. true: sort as strings.
- `descending` — true: sort from largest to smallest.
- `caseInsensitive` — true: ignore case when sorting as strings.

Unlike JavaScript, `sort()` without arguments sorts **numerically**, and it compares only the whole-number part: 1.2 and 1.7 are treated as equal. Values that are not numbers are treated as 0. To sort decimals exactly, use a compare function. To sort text, pass `true`.

```
print([10, 2, 33, 4].sort(), "\n");
print([3, 1, 2].sort(false, true), "\n");
print(["b", "C", "a"].sort(true), "\n");
print(["b", "C", "a"].sort(true, false, true), "\n");
print([1.7, 1.2, 0.5].sort(function (a, b) { return a - b; }), "\n");
people = [{"name": "Ann", "age": 41}, {"name": "Bob", "age": 25}];
people.sort(function (a, b) { return a.age - b.age; });
return people[0].name;
```

Print output:

```
[2, 4, 10, 33]
[3, 2, 1]
[C, a, b]
[a, b, C]
[0.5, 1.2, 1.7]
```

Output: Bob

forEach, where, map

These methods call a function once for each item. The function receives **only the item**; there is no index argument as in JavaScript.

Method	Returns
list.forEach(function)	null. Use it for side effects
list.where(function)	A new list of the items for which the function returns a true value (JavaScript filter)
list.map(function)	A new list of the values returned by the function

There is no filter, reduce, find, some, every, includes or reverse. Use where, a loop, or indexOf.

```
nums = [1, 2, 3, 4, 5];
total = 0;
nums.forEach(function (n) { total += n; });
print(total, "\n");
print(nums.where(function (n) { return n % 2 == 0; }), "\n");
return nums.map(function (n) { return n * 10; });
```

Print output:

```
15
[2, 4]
```

Output: [10, 20, 30, 40, 50]

Maps

Create a map with curly braces. Keys are names or quoted strings. Read and write values with a dot or with square brackets.

```
pump = {name: "Pump 1", "max speed": 3000};
pump.status = "on";
pump["max speed"] = 3600;
print(pump.name, " ", pump["max speed"], " ", pump.status, "\n");
print(pump.missing, "\n");
print("status" in pump, "\n");
delete pump.status;
print("status" in pump, "\n");
print(typeof pump, "\n");
return pump;
```

Print output:

```
Pump 1 3600 on
null
true
false
object
```

Output: {name: Pump 1, max speed: 3600}

Things to know:

- Reading a key that does not exist returns `null`.
- `key in map` returns `true` if the key exists, even when its value is `null`.
- `delete map.key` removes the key.
- Keys keep the order in which they were added.
- Keys are always strings. **Numeric keys do not work** in a map literal or as a bracket key when writing: `{1: "a"}` creates an empty map, and `m[1] = "a"` does nothing. Use string keys: `{"1": "a"}` and `m[String(1)] = "a"`. Reading with `m[1]` works and reads the key "1".
- A map has no `length` and no methods such as `keys()`. To count or list the keys, use [for...in](#).
- Use `JSON.stringify` to turn a map into a JSON string. See [JSON and XML](#).

Looping with for...in

for (key in map) runs the loop once for each key of a map. for (i in list) runs once for each **position** (0, 1, 2, ...) of a list, not for each item. break and continue work as usual.

```
m = {a: 1, b: 2, c: 3};
keys = [];
for (k in m) {
    keys.push(k + "=" + m[k]);
}
print(keys.join(", "), "\n");

list = ["x", "y", "z"];
for (var i in list) {
    print(i, ":", list[i], " ");
}
return keys.length;
```

Print output:

```
a=1,b=2,c=3
0:x 1:y 2:z
```

Output: 3

Array(value)

Array(value) creates or converts a list:

- Array(n) with a whole number n returns a list of n null items. This list has a **fixed length**: you can set items but cannot push, pop or change length. Use [] and push when you need a growable list.
- Array(list) returns the same list.
- Array(binaryData) returns the bytes of binary data as a list of numbers.
- Array(modelArray) converts an array stored in the page model to a list.
- Any other value returns null.

There is no new Array() syntax.

```
a = Array(3);
a[0] = "first";
```

```
print(a, "\n");  
return a.length;
```

Print output:

```
[first, null, null]
```

Output: 3

From:

<http://wiki.dglogik.com/> - **DGLogik**

Permanent link:

http://wiki.dglogik.com/dglux5_wiki:dgscript:lists_maps

Last update: **2026/10/22 17:28**

