# THE Things You Can't Do

- Snowboard
- Fly the Space Shuttle
- Have NaN equal something else.
- Find solace amongst all this despair.

# THE Things You Can't Do

## Loopy-Doop

```
var items = [1, 2, 3];
var items = [...];
var items = [...];

for(var i = 0;
    i < items.length;
    i++) {
  console.log(items[i]);
}
```

```
var items = [...];

for(var i = 0;
    i < items.length;
    i++) {
  console.log(items[i]);
}
```

```
var items = [...];

for(var i = 0;
    i < items.length;
    i++) {
  console.log(items[i]);
}
```

```
var items = [...];

for(var i = 0;
    i < items.length;
    i++) {
  console.log(items[i]);
}
```

```
var items = [...];

for(var i = 0;
    i < items.length;
    i++) {
  console.log(items[i]);
}
```

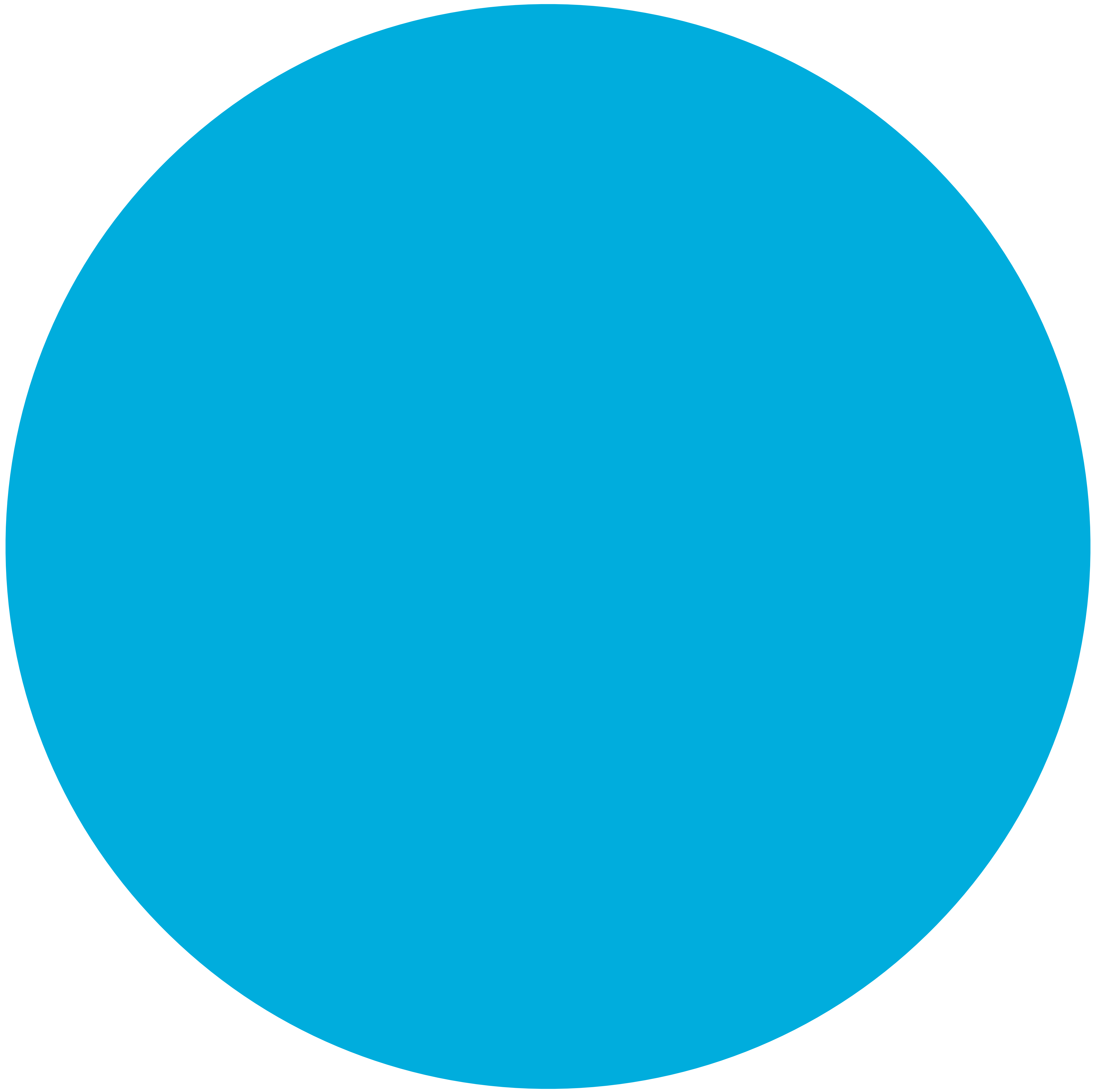
```
var items = [1,2,3];

for(var i = 0;
    i < items.length;
    i++) {
  console.log(items[i]);
}
// 1  2  3
```

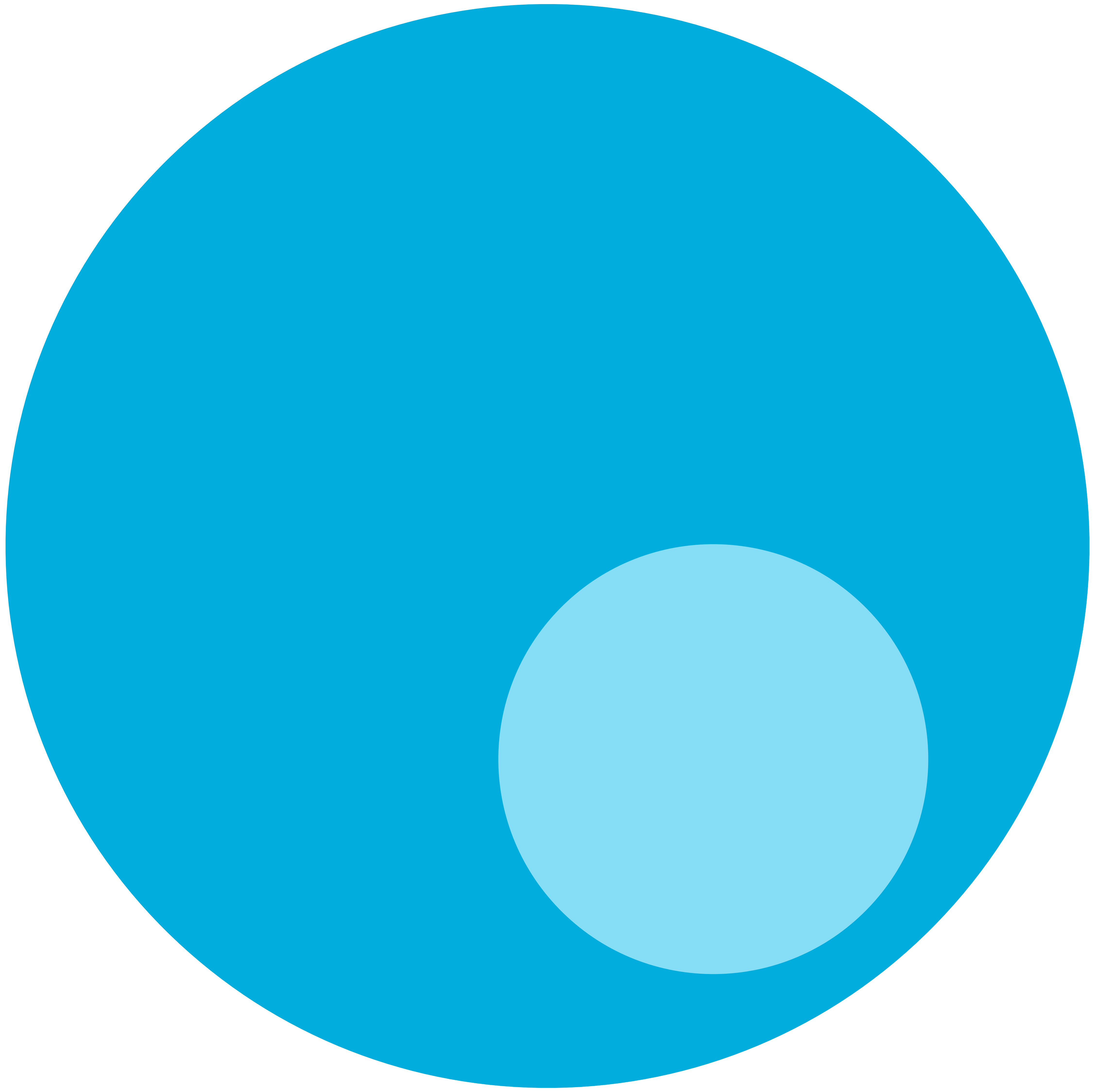
```
var items = [1,2,3];

for(var i = 0;
    i <= items.length;
    i++) {
  console.log(items[i]);
}

// 1  2  3  undefined
```

```
var items = [1,2,3];

for(var i = 1;
    i < items.length;
    i++) {
  console.log(items[i]);
}

// 2  3
```

```
var items = [1,2,3];

for(var i = 0;
    i < items.length;
    i++) {
  console.log(items[i]);
}
// 1  2  3
```

```
var items = [1,2,3];

for(var i=items.length;
      i > 0;
      i--) {
  console.log(items[i]);
}

// 3  2  1
```



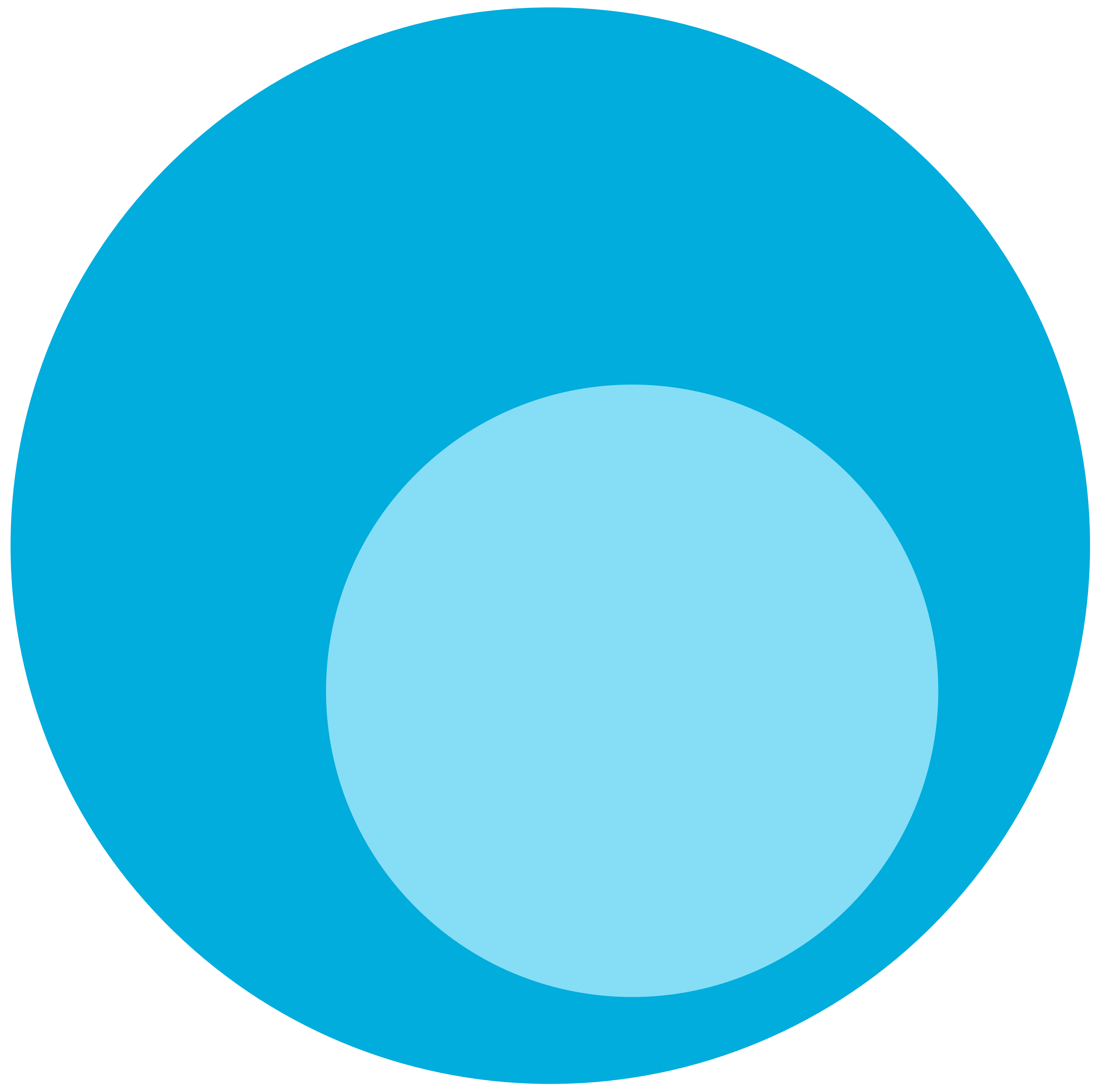


```
var items = [...];

for(var i in items) {
  console.log(items[i]);
}
```

```
var items = [1,2,3];

for(var i in items) {
  console.log(items[i]);
}
// 1
// 2
// 3
```

```
var items =
 document.querySelectorAll('p');
for(var i in items) {
  console.log(items[i]);
}

// <p>...</p>
// <p>...</p>
// function item()
// 4
```

```
var items =
 document.querySelectorAll('p');
for(var i in items) {
  if (items.hasOwnProperty(i)) {
    console.log(items[i]);
  }
}

// <p>...</p>
// <p>...</p>
```



```
var items = [1,2,3];

items.forEach(function(x){
  console.log(x);
});

// 1
// 2
// 3
```

```
var items = [1,2,3];

items.forEach(function(x){
  console.log(x);
});

// 1
// 2
// 3
```

```
var items = [1,2,3];
var added = [];
items.forEach(function(x){
  added.push(x + 1);
});
console.log(added);
// [ 2, 3, 4 ]
```

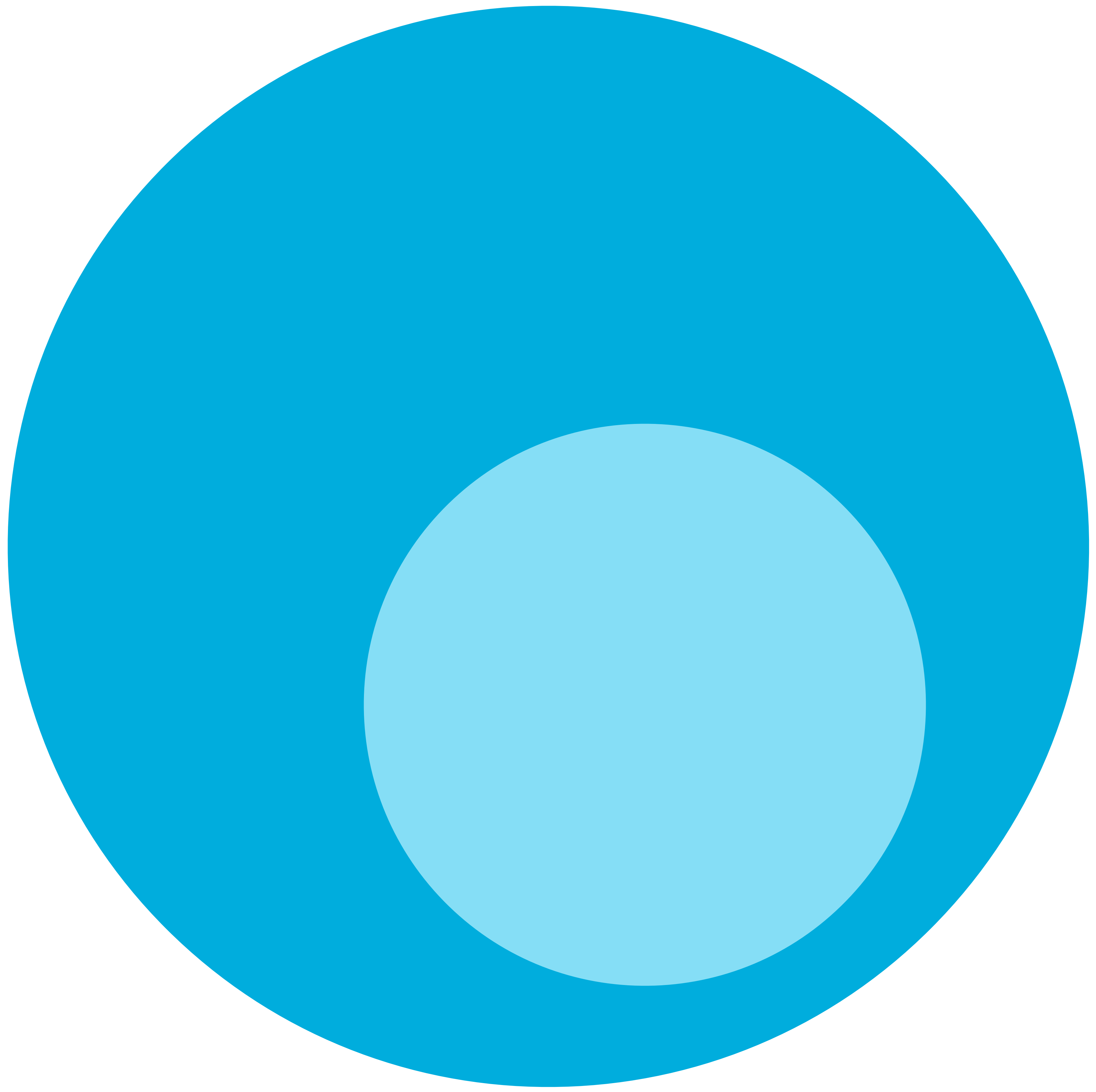
```
var items = [1,2,3];
var added = [];
items.forEach(function(x){
  added.push(x + 1);
});
console.log(added);
// [ 2, 3, 4 ]
```

```
var items = [1,2,3];
var added = [];
items.forEach(function(x){
  added.push(x + 1);
});
console.log(added);
// [ 2, 3, 4 ]
```

```
var items = [1,2,3];
var added = [];
items.forEach(function(x){
  added.push(x + 1);
});
console.log(added);
// [ 2, 3, 4 ]
```

```
var items = [1,2,3];
var added = [];
items.forEach(function(x){
  added.concat(x + 1);
});
console.log(added);
// []
```

```
var items = [1,2,3];
var added = [];
items.forEach(function(x){
  items.push(x + 1);
});
console.log(added);
// []
console.log(items);
// [ 1, 2, 3, 2, 4, 5 ]
```

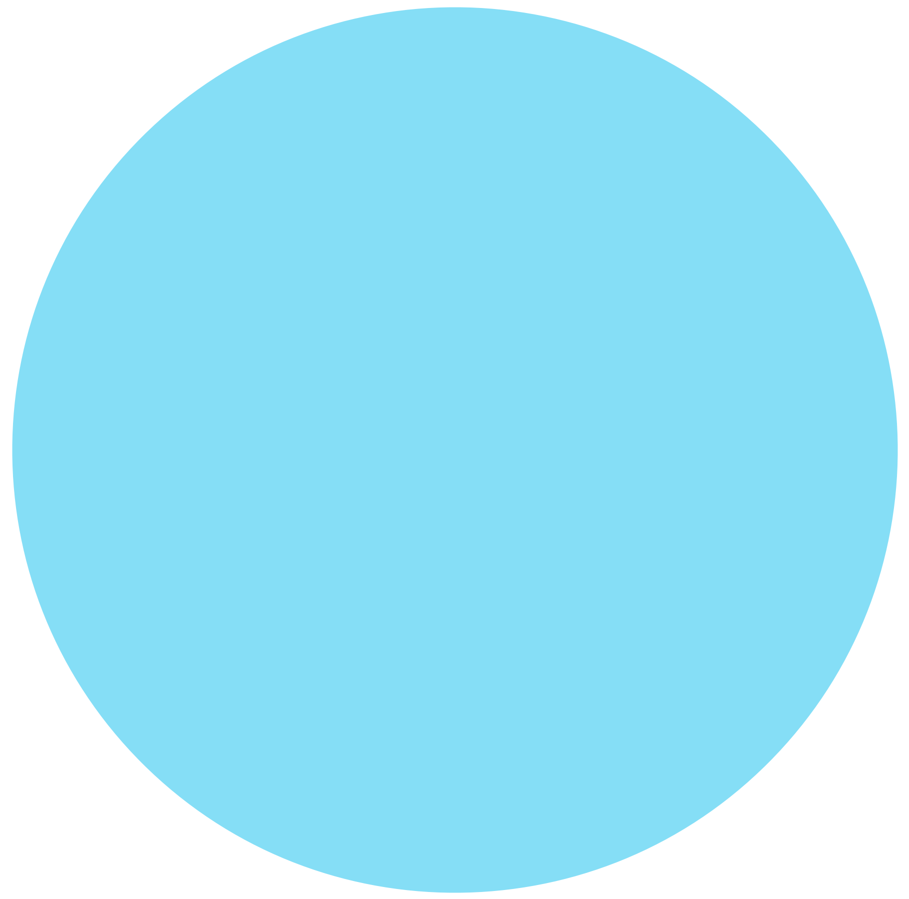


```
var items = [1,2,3];
var added = [];
items.forEach(function(x){
  added.push(x + 1);
});

console.log(added);
// [ 2, 3, 4 ]
```

```
var items = [1,2,3];
var added =
  items.map(function(item){
    return item + 1;
  });

console.log(added);
// [ 2, 3, 4 ]
```

```
var items = [1,2,3];
var added =
  items.map(function(item){
    return item + 1;
  });

console.log(added);
// [ 2, 3, 4 ]
```

```
var items = [1,2,3];
var added =
  items.map(function(item){
    return item + 1;
  });

console.log(added);
// [ 2, 3, 4 ]
```



```
var items = [1,2,3];
var evenNumbers =
  items.filter(
    function(item){
      return item % 2 === 0;
    }
  );
console.log(evenNumbers);
// [2]
```

```
var items = [1,2,3];
var sum =
  items.reduce(
    function(total, item){
      return total + item;
    }, 0
  );
console.log(sum);
// 6
```

forEach
map
reduce
filter
etc.

## ES6

## ES2015

## ES2099

# ES40000

```
var items = [1,2,3];

for(var i = 0;
    i < items.length;
    i++) {
  console.log(items[i]);
}
```

```
var items = [1,2,3];

for(var i = 0;
    i < items.length;
    i++) {
  console.log(items[i]);
}
console.log(i);
// 3
```

```
var items = [1,2,3];

for(let i = 0;
    i < items.length;
    i++) {
  console.log(items[i]);
}
console.log(i);
```

```
var items = [1,2,3];

for(let i = 0;
    i < items.length;
    i++) {
  console.log(items[i]);
}
console.log(i);
// 💥 i is not defined.
```

```
const i = 1;

// ... later ...

i = 2; // 💥
```

let

const

## Immutability

```
var evens =
  evenNumbers(bigList(12));
  // [2,4,6,8,10,12]

var tens =
  endsInZero(bigList(12));
  // [10]
```

```
var evens =
  evenNumbers(bigList(12));
  // [2,4,6,8,10,12]

var tens =
  endsInZero(bigList(12));
  // [10]
```

```
var list = bigList(12);
var evens =
  evenNumbers(list);
  // [2,4,6,8,10,12]

var tens =
  endsInZero(list);
  // []
```

```
var list = bigList(12);
var evens =
  evenNumbers(list);
  // [2,4,6,8,10,12]

var tens =
  endsInZero(list);
  // []💦
```

```
function evenNumbers(list) {
  var result = []
  while (list.length > 0) {
    var num = list.shift();
    // Changes the array!
    if (num % 2 == 0) {
      result.push(num)
    }
  }
  return result;
}
```

```
function evenNumbers(list) {
  var result = []
  while (list.length > 0) {
    var num = list.shift();
    // Changes the array!
    if (num % 2 == 0) {
      result.push(num)
    }
  }
  return result;
}
```

```
var list =
  bigList(12);
var evens =
  evenNumbers(list);
  // [2,4,6,8,10,12]
var tens =
  endsInZero(list);
  // []
```

```
var Immutable =
  require('seamless-immutable');
var list =
  Immutable(bigList(12));

var evens =
  evenNumbers(list);
  // ...
var tens =
  endsInZero(list);
  // ...
```

```
var Immutable =
  require('seamless-immutable');
var list =
  Immutable(bigList(12));

var evens =
  evenNumbers(list);
  // 💥 ImmutableError: The
  // shift method cannot be
  // invoked on an Immutable
  // data structure.
```

```
var list =
  Object.freeze(bigList(12));

var evens =
  evenNumbers(list);
  // 💥 TypeError: Cannot
  // add/remove sealed array
  // elements
```

```
var list =
  Object.freeze([{a:123},{a:456}]);

list[0].a = 999;
console.log(list[0].a);
// 999
```

seamless-immutable

Immutable.js

## The Kinds of Things

```
var backwards =
  myUtils.reverseStr(
    undefined
  );
```

```
var backwards =
  myUtils.reverseStr(
    undefined
  );
// TypeError: Cannot read
// property 'split' of undefined
//    at reverse (...)
```

```
var backwards =
  myUtils.reverseStr(
    7
  );
```

```
var backwards =
  myUtils.reverseStr(
    7
  );
// TypeError: x.split is not a function
//    at reverse (...)
```

```
function (x) {
  var backwards =
    myUtils.reverseStr(
      x
    );
  // ...
}
```

```
function (x) {
  var backwards =
    myUtils.reverseStr(
      x
    );
  // ...
}
// ...?
```

```
function reverseStr(x) {
  return x.split("")
          .reverse()
          .join("");
}
```

```
function reverseStr(x) {
  return x.split("")
          .reverse()
          .join("");
}
```

```
function reverseStr(
  x: string
): string {
  return x.split("")
          .reverse()
          .join("");
}
```

```
function reverseStr(
  x: string
): string {
  return x.split("")
          .reverse()
          .join("");
}
reverseStr("hello"); // 👍
```

```
function reverseStr(
  x: string
): string {
  return x.split("")
          .reverse()
          .join("");
}
reverseStr(7);       // 💥
```

```
function reverseStr(
  x: string
): string {
  return x.split("")
          .reverse()
          .join("");
}
reverseStr([1,2,3]); // 💥
```

```
$ cat example.js
$ cat example.js
// @flow
function reverseStr(x: string): string {
  return x.split("").reverse().join("");
}
console.log(reverseStr("hello"));
```

```
$ cat example.js
// @flow
function reverseStr(x: string): string {
  return x.split("").reverse().join("");
}
console.log(reverseStr("hello"));

$ flow check
$ cat example.js
// @flow
function reverseStr(x: string): string {
  return x.split("").reverse().join("");
}
console.log(reverseStr("hello"));

$ flow check
Found 0 errors
```

```
$ cat example.js
// @flow
function reverseStr(x: string): string {
  return x.split("").reverse().join("");
}
console.log(reverseStr("hello"));

$ flow check
Found 0 errors

$ babel-node example.js
$ cat example.js
// @flow
function reverseStr(x: string): string {
  return x.split("").reverse().join("");
}
console.log(reverseStr("hello"));

$ flow check
Found 0 errors

$ babel-node example.js
olleh
```

```
$ cat example.js
// @flow
function reverseStr(x: string): string {
  return x.split("").reverse().join("");
}
console.log(reverseStr(987));
```

```
$ cat example.js
// @flow
function reverseStr(x: string): string {
  return x.split("").reverse().join("");
}
console.log(reverseStr(987));

$ flow check
example.js:5
  5: console.log(reverseStr(987));
                 ^^^^^^^^^^^^^^^ function call
  5: console.log(reverseStr(987));
                            ^^^ number.
     This type is incompatible with
  2: function reverseStr(x: string): string {
                            ^^^^^^ string

Found 1 error
```

```
function reverseStr(
  x: string
): string {
  return x.split("")
          .reverse()
          .join("");
}
```

```
function sort(
  x: Array<number>
): Array<number> {
  return ...
}
```

```
function sort(
  x: Array<number>
): Array<number> {
  return ...
}


sort([3,7,1]);  // 👍
```

```
function sort(
  x: Array<number>
): Array<number> {
  return ...
}


sort("hi");     // 💥
```

```
function reverse<T>(
  x: Array<T>
): Array<T> {
  return ...
}
```

```
function reverse<T>(
  x: Array<T>
): Array<T> {
  return ...
}


reverse([1,2,3]);    // 👍
```

```
function reverse<T>(
  x: Array<T>
): Array<T> {
  return ...
}


reverse(['a','b']);  // 👍
```

```
function reverse<T>(
  x: Array<T>
): Array<T> {
  return ...
}


reverse([true]);     // 👍
```

```
function greet(
  x: {username: string}
): string {
  return "Hi, " + x.username;
}

greet(
 {username: "Jean", id: 24601}
); // 👍
```

```
function defaultTo(
  d: string, val?: string
): string {
  if (val === undefined)
    return d;
  else
    return val;
}

defaultTo("brave", adjective); // 👍
```

```
type Thing = boolean | number;
function thingToString(
  x: Thing
): string {
  return ...
}
```

```
type Action =
    { type: "LOGGED_IN", user: string }
  | { type: "LOGGED_OUT" }

function user(
  state: State, action: Action
): State {
  if (action.type === "LOGGED_IN") {
    return {name: action.user, isLoggedIn: true}
  }
  if (action.type === "LOGGED_OUT")
  ...
}
```

```
type Action =
    { type: "LOGGED_IN", user: string }
  | { type: "LOGGED_OUT" }

function user(
  state: State, action: Action
): State {
  if (action.type === "LOGGED_IN") {
    return {name: action.user, isLoggedIn: true}
  }
  if (action.type === "LOGGED_OUT")
  ...
}
```

```
type Action =
    { type: "LOGGED_IN", user: string }
  | { type: "LOGGED_OUT" }

function user(
  state: State, action: Action
): State {
  if (action.type === "LOGGED_IN") {
    return {name: action.user, isLoggedIn: true}
  }
  if (action.type === "LOGGED_OUT")
  ...
}
```

```
type Action =
    { type: "LOGGED_IN", user: string }
  | { type: "LOGGED_OUT" }

function user(
  state: State, action: Action
): State {
  if (action.type === "LOGGED_IN") {
    return {name: action.user, isLoggedIn: true}
  }
  if (action.type === "LOGGED_OUT")
  ...
}
```

```
type Action =
    { type: "LOGGED_IN", user: string }
  | { type: "LOGGED_OUT" }

function user(
  state: State, action: Action
): State {
  if (action.type === "LOGGED_IN") {
    return {name: action.user, isLoggedIn: true}
  }
  if (action.type === "LOGGED_OUT")
  ...
}
```

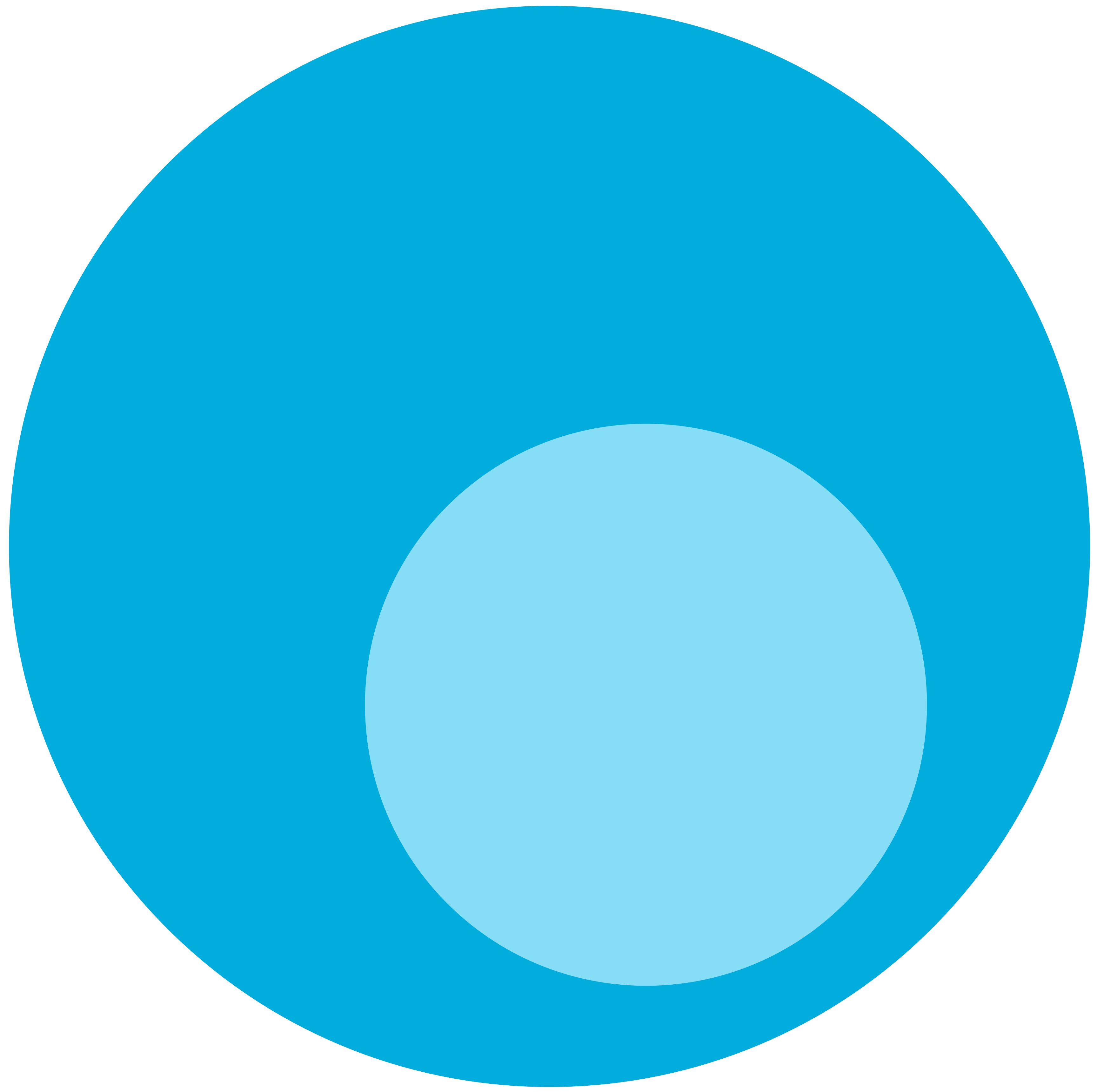
```
function reverseStr(
  x: string
): string {
  return x.split("")
          .reverse()
          .join("");
}
```

```
function reverseStr(x) {
  return x.split("")
          .reverse()
          .join("");
}
```

```
function add(x, y) {
  return x + y;
}
```

```
function add(x, y) {
  return x + y;
}

add(1, 2)      // 👍
```

```
function add(x, y) {
  return x + y;
}

add("abc", 2) // 👍❓😰
```



```
function add(x, y) {
  return x + y;
}

add("abc", 2) // 👍❓😰
```

```
function add(
  x: number, y: number
): number {
  return x + y;
}

add(1, 2)     // 👍
add("abc", 2) // 💥
```

Flow

TypeScript

## Fit for Purpose

## Rule of Least Power

## JSON

## HTML

## CSS

# SQL

## Queries, Inserts, Updates

## forEach()

## map()

let

## const

## Immutable(...)

reverse(x:string):string

⬇️

# THE Things You Can't Do

THE

# Things You Can't Do

robhoward.id.au
@damncabbage

# THE Things You Can't Do

tinyurl.com/wdc16-cantdo

robhoward.id.au
@damncabbage