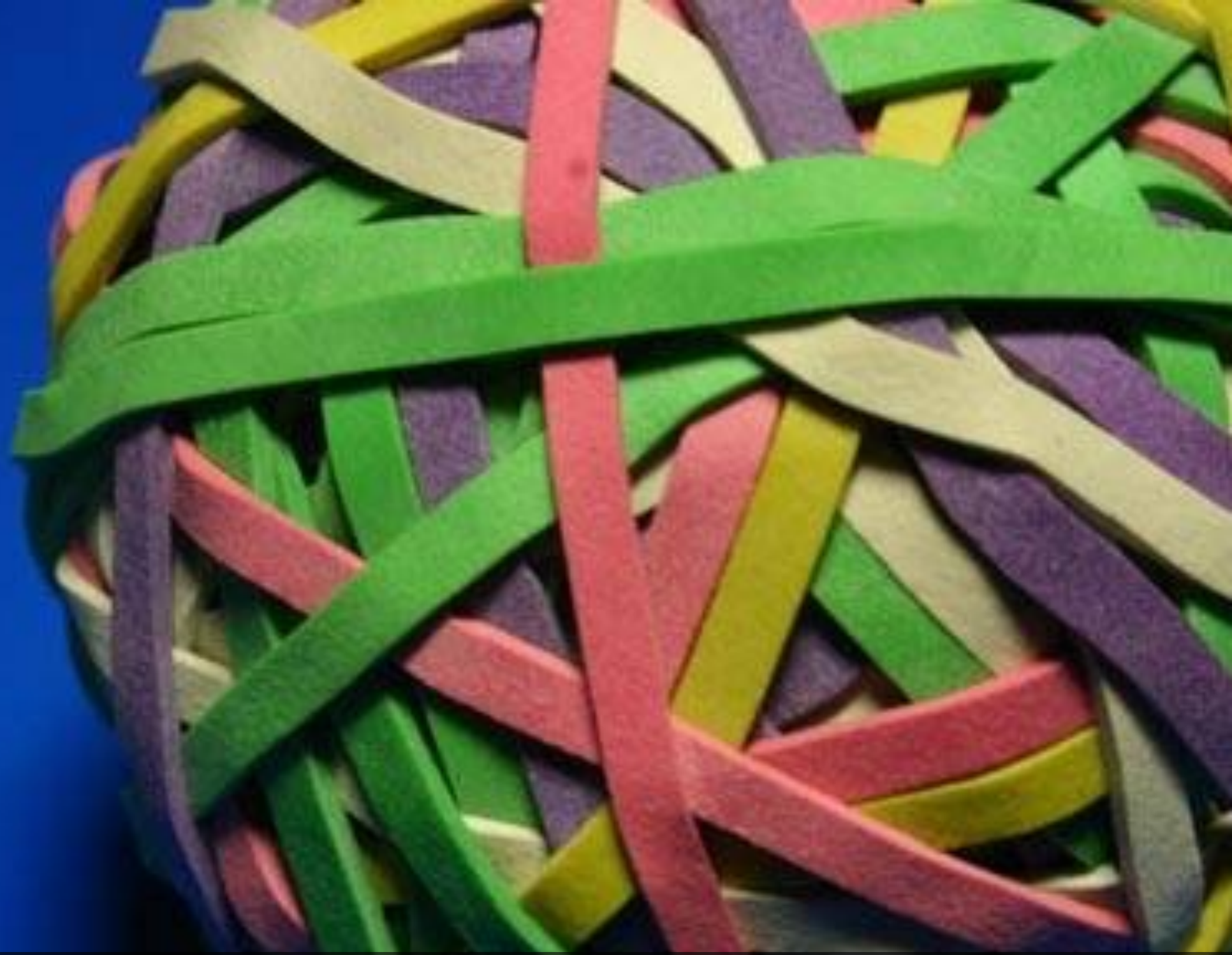

Closures

Na linguagem JavaScript, toda função permite a utilização de variáveis que não foram declaradas e nem passadas por parâmetro

closures_1.js — javascriptmasterclass

closures_1.js

```
1  const v1 = 10;
2  function fn1() {
3      console.log(v1);
4  };
5  fn1();
6
```

TERMINAL 1: bash

```
rodrigobranas:javascriptmasterclass $ node c
losures/closures_1.js
10
rodrigobranas:javascriptmasterclass $
```

closures_2.js — javascriptmasterclass

closures_2.js

```
1  const v1 = 10;
2  function fn1() {
3      function fn2() {
4          console.log(v1);
5      }
6      fn2();
7  };
8  fn1();
9
```

TERMINAL 1: bash

```
rodrigobranas:javascriptmasterclass $ node c
losures/closures_2.js
10
rodrigobranas:javascriptmasterclass $
```

O problema é que como as funções são de primeira classe, dependendo da situação poderia existir uma ambiguidade e isso foi resolvido com o conceito de closure

closures_3.js — javascriptmasterclass

closures_3.js

```
1  function fn1() {
2      const v1 = 10;
3      return function() {
4          console.log(v1);
5      };
6  };
7  const fn2 = fn1();
8  fn2();
9
```

TERMINAL 1: bash

```
rodrigobranas:javascriptmasterclass $ node c
losures/closures_3.js
10
rodrigobranas:javascriptmasterclass $
```

closures_4.js — javascriptmasterclass

closures_4.js

```
1  function fn1() {
2      const v1 = 10;
3      return function() {
4          console.log(v1);
5      };
6  };
7  const fn2 = fn1();
8  const v1 = 100;
9  fn2();
10
```

TERMINAL

1: bash

```
rodrigobranas:javascriptmasterclass $ node c
losures/closures_4.js
10
rodrigobranas:javascriptmasterclass $
```

Resumindo, closure é uma função com um scope chain estático que é definida no momento em que a função é criada, por isso, todas as funções na linguagem JavaScript são closures

closures_5.js — javascriptmasterclass

closures_5.js

```
1   const v1 = 10;
2   function fn1() {
3       console.log(v1);
4   }
5   function fn2(fn1) {
6       const v1 = 100;
7       fn1();
8   }
9   fn2(fn1);
10
```

TERMINAL 1: bash

```
rodrigobranas:javascriptmasterclass $ node c
losures/closures_5.js
10
rodrigobranas:javascriptmasterclass $
```

Apesar de estático, o scope chain faz referência para objetos que estão na memória e podem ser compartilhados por mais de uma função

closures_6.js — javascriptmasterclass

closures_6.js

```
1   function fn1() {
2       let v1 = 10;
3       return {
4           m1() {
5               console.log(++v1);
6           },
7           m2() {
8               console.log(--v1);
9           }
10      };
11  };
12  const obj1 = fn1();
13  obj1.m1();
14  obj1.m2();
15
```

TERMINAL

1: bash

```
rodrigobranas:javascriptmasterclass $ node c
losures/closures_6.js
11
10
rodrigobranas:javascriptmasterclass $
```

closures_7.js — javascriptmasterclass

closures_7.js

```
1   var obj1 = {};
2   for (var v1 = 0; v1 < 3; v1++) {
3     obj1[v1] = function () {
4       console.log(v1);
5     };
6   }
7   obj1[0]();
8   obj1[1]();
9   obj1[2]();
10
```

TERMINAL 1: bash

```
rodrigobranas:javascriptmasterclass $ node c
losures/closures_7.js
3
3
3
rodrigobranas:javascriptmasterclass $
```

closures_8.js — javascriptmasterclass

closures_8.js

```
1   var obj1 = {};
2   for (var v1 = 0; v1 < 3; v1++) {
3     obj1[v1] = (function (v2) {
4         return function () {
5           console.log(v2);
6         };
7     })(v1);
8   }
9   obj1[0]();
10  obj1[1]();
11  obj1[2]();
12
```

TERMINAL

1: bash

```
rodrigobranas:javascriptmasterclass $ node c
losures/closures_8.js
0
1
2
rodrigobranas:javascriptmasterclass $
```

closures_9.js — javascriptmasterclass

closures_9.js

```
1   var obj1 = {};
2   for (var v1 = 0; v1 < 3; v1++) {
3     obj1[v1] = (function () {
4       console.log(this.v2);
5     }).bind({v2: v1});
6   }
7   obj1[0]();
8   obj1[1]();
9   obj1[2]();
10
```

TERMINAL 1: bash

```
rodrigobranas:javascriptmasterclass $ node c
losures/closures_9.js
0
1
2
rodrigobranas:javascriptmasterclass $
```