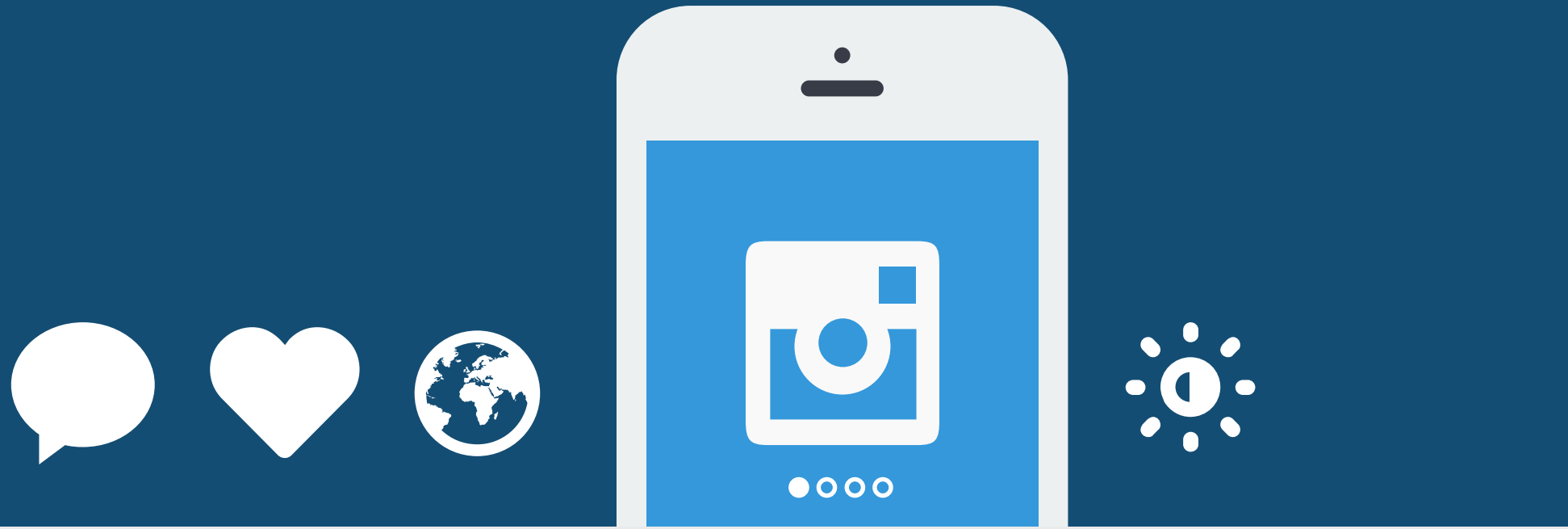
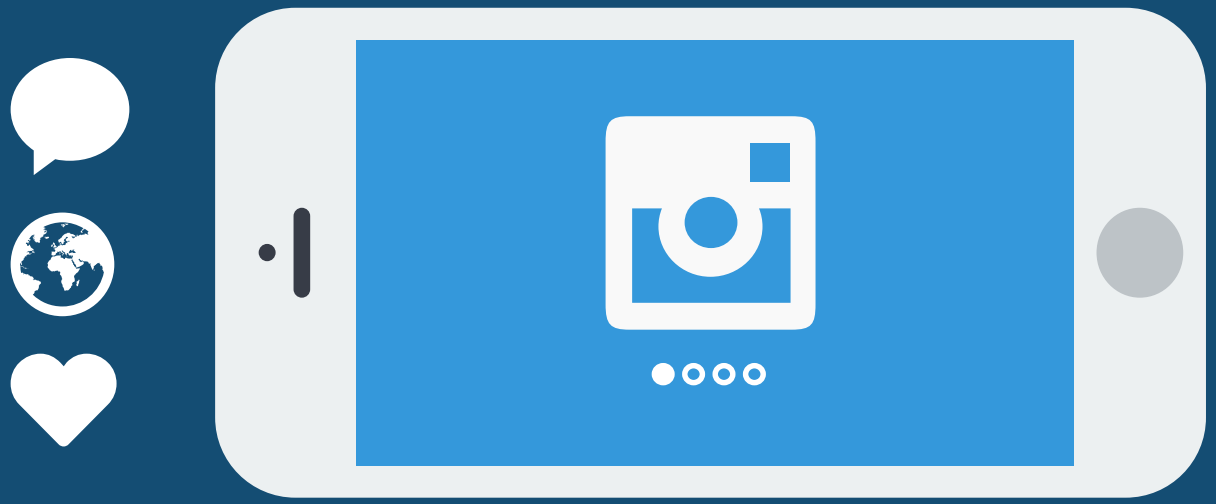
targettrust

treinamento e tecnologia



FUNÇÕES DE ALTA ORDEM



Funções de Alta Ordem

- Um software grande é um software de alto custo e não apenas pelo tempo que leva para construir.

- Tamanho quase sempre envolve complexidade e complexidade confunde programadores.

- Programadores confusos tendem a inserir bugs no software. Um software grande é um bom lugar para esconder bugs.

Funções de Alta Ordem

- Qual dos softwares é mais propenso à bugs?
- Porquê?

```
var total = 0, count = 1;
while (count <= 10) {
        total += count;
        count += 1;
}
console.log(total);
console.log(sum(range(1, 10)));
```

Funções de Alta Ordem

- O segundo código é menos propenso a erros.

- Além disso, oferece um maior nível de abstração. A abstração é positiva para simplicidade, mas exige um conhecimento maior do programador.

Funções de Alta Ordem

- Um tipo de função comum. O que ela faz?
- Quais são os erros possíveis e comuns neste trecho de código?

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

Funções de Alta Ordem

Exercício 01:

- Escreva uma função maximizando o reuzo do loop. **Dica**: use ao menos um parâmetro.

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

Funções de Alta Ordem

- E se eu quiser fazer outras coisas além printar no console?

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


Funções de Alta Ordem

```
function forEach(array, action) {
        for (var i = 0; i <
        array.length; i++)
            action(array[i]);
        }
var numbers = [1, 2, 3, 4, 5], sum
= 0;
forEach(numbers,
function(number) {
        sum += number;
});
console.log(sum);
```


Funções de Alta Ordem

```
function gatherCorrelations(journal) {
        var phis = {};
        for (var entry = 0; entry < journal.length; entry++) {
            var events = journal[entry].events;
            for (var i = 0; i < events.length; i++)
```

```
{
                var event = events[i];
                if (!(event in phis))
                    //phis[event] = phi(tableFor(event, journal));
            }
        }
        return phis;
}
```

Funções de Alta Ordem

- Arrays javascript possuem métodos embutidos para executar for each.

Funções de Alta Ordem

```
function gatherCorrelations(journal) {
    var phis = {};
    journal.forEach(function(entry) {
        entry.events.forEach(function(event) {
            if (!(event in phis))
                phis[event] = phi(tableFor(event, journal));
        });
    });
    return phis;
}
```

Funções de Alta Ordem

- Funções que operam dentro de outras funções são chamadas “Alta-Ordem” (High-Order).
- Funções de alta-ordem nos permite abstrair a partir de ***ações*** além de valores.
- Funções podem criar ou alterar outras funções.
- Exercício 02: complete o código.

Funções de Alta Ordem

```
function maiorQue(n) {
        return _____________________;
}
var maiorQue10 = maiorQue(10);
console.log(maiorQue10(11));
```

Funções de Alta Ordem

```
function maiorQue(n) {
        return function(m) { return m > n; };
}
var maiorQue10 = maiorQue(10);
console.log(maiorQue10(11));
```

Funções de Alta Ordem

Exercício 03:

- Complete o código a fim de obter um retorno igual a: ***chamado com 0 - obitve false.***

Funções de Alta Ordem

```
function noisy(f) {
    return function(arg) {
        console.log("chamando com", arg);
        var val = f(arg);
        console.log("chamado com", arg, "- obtive", val);
        return val;
    };
}
```

Funções de Alta Ordem

```
function noisy(f) {
    return function(arg) {
        console.log("chamando com", arg);
        var val = f(arg);
        console.log("chamado com", arg, "- obtive", val);
        return val;
    };
}
noisy(Boolean)(0);
```

Funções de Alta Ordem

Exercício 04:

- A função a seguir exibe o quê na tela?

Funções de Alta Ordem

```
function aMenosQue(teste, entao) {
        if (!teste) entao();
}
function repita(vezes, body) {
        for (var i = 0; i < vezes; i++)
        body(i);
}
repita(3, function(n) {
        aMenosQue(n % 2, function() {
            console.log(n, "é par");
        });
});
```

Funções de Alta Ordem

- Note que: as *inner functions* conseguem utilizar as variáveis declaradas “ao redor” delas. Escopo similiar aos { } de um laço.

- Entretanto, as variáveis das *inner functions* não podem ser utilizadas pelas *outer functions*. Geralmente, isto é uma característica positiva.

Do quê se Trata?

[
{"name": "Emma de Milliano", "sex": "f",
"born": 1876, "died": 1956,
"father": "Petrus de Milliano",
"mother": "Sophia van Damme"},
{"name": "Carolus Haverbeke", "sex": "m",
"born": 1832, "died": 1905,
"father": "Carel Haverbeke",
"mother": "Maria van Brussel"}
]

JSON

JSON

- JavaScript Object Notation
- Sem funções
- Sem comentários
- Sem variáveis
- JSON.stringify e JSON.parse

JSON

```
var string = JSON.stringify({name: "X", born: 1980});
console.log(string);
console.log(JSON.parse(string).born);
```

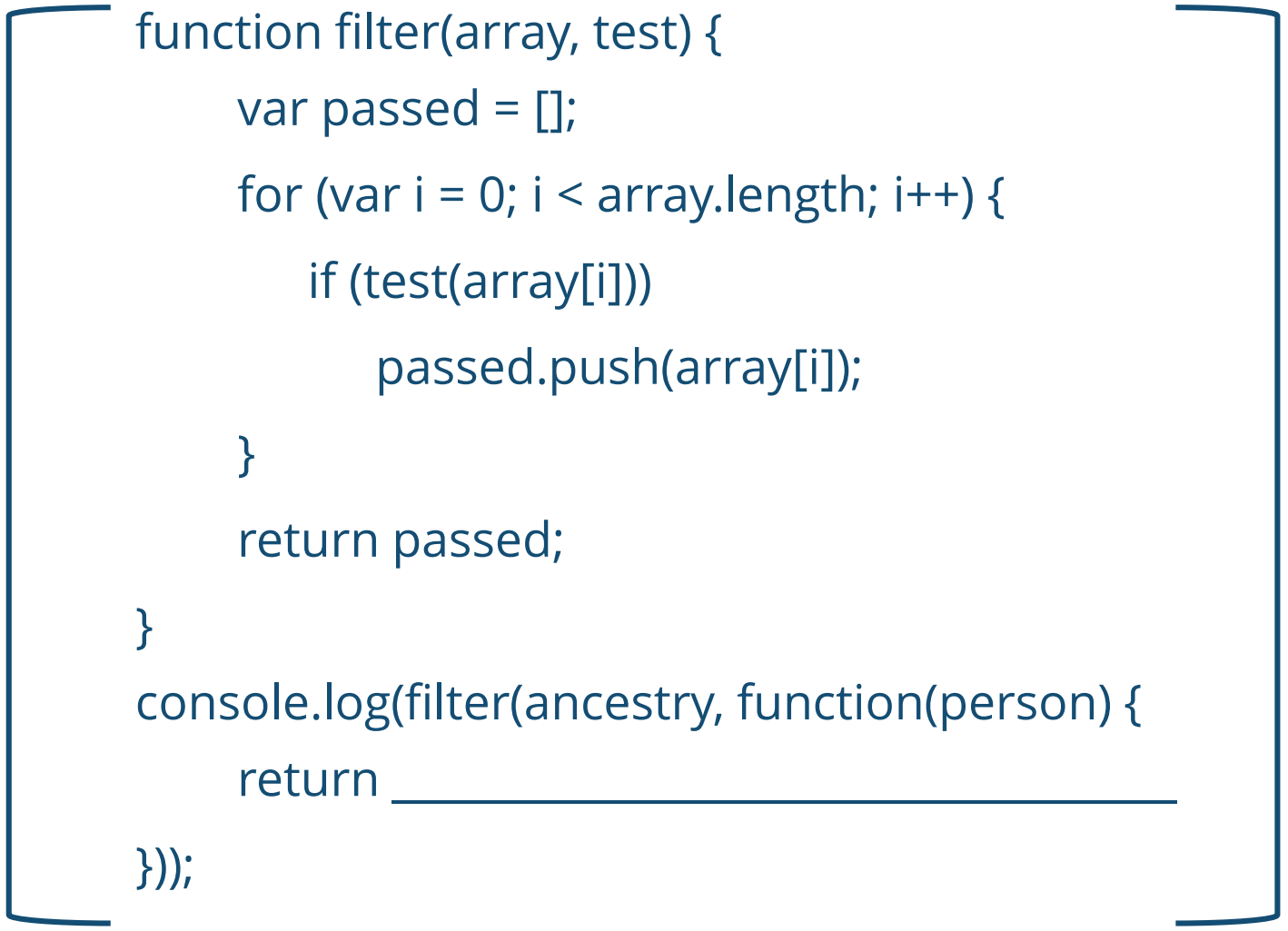
JSON

- http://eloquentjavascript.net/code/ancestry.js

```
var ancestry = JSON.parse(ANCESTRY_FILE);
console.log(ancestry.length);
```

- Qual número é printado no console?
- Exercício 05: Quais ancestrais eram jovens em 1924 (1 a 25 anos)?

JSON



```
function filter(array, test) {
        var passed = [];
        for (var i = 0; i < array.length; i++) {
            if (test(array[i]))
                passed.push(array[i]);
        }
        return passed;
}
console.log(filter(ancestry, function(person) {
        return _____
}));
```

JSON

```
function filter(array, test) {
        var passed = [];
        for (var i = 0; i < array.length; i++) {
            if (test(array[i]))
                passed.push(array[i]);
        }
        return passed;
}
console.log(filter(ancestry, function(person) {
        return return person.born > 1900 &&
            person.born < 1925;
}));
```

JSON

- Boa notícia: filter também é uma função disponibilizada pelo Javascript. E sem mexer no vetor.

```
console.log(ancestry.filter(function(person) {
        return person.father == "Carel Haverbeke";
        }))
```

JSON

- Se tivermos um vetor de objetos representando pessoas através do filtro do vetor de ancestrais.
- Mas queremos um vetor de nomes, que é mais fácil de ler.

JSON

- O método ***map*** transforma um vetor aplicando uma função em todos os seus elementos e construindo um novo vetor dos valores retornados.

- O novo vetor tem o mesmo tamanho do vetor de entrada, mas o seu conteúdo terá sido mapeado para um novo formato pela função.

- **PS: complete o código**

JSON

```
function map(array, transform) {
    var mapped = [];
    for (var i = 0; i <
    array.length; i++)
        mapped.push(transform(
        array[i]));
    return mapped;
}
var overNinety =
ancestry.filter(function(person) {

_____

});
```

```
console.log(map(overNinety,
function(person) {
return person.name;
}));
```

JSON

```
function map(array, transform) {
        var mapped = [];
        for (var i = 0; i <
        array.length; i++)
            mapped.push(transform(
            array[i]));
        return mapped;
}
var overNinety =
ancestry.filter(function(person) {
        return person.died -
            person.born > 90;
});
```

```
console.log(map(overNinety,
function(person) {
return person.name;
}));
```

JSON

```
function reduce(array, combine,
start) {
    var current = start;
    for (var i = 0; i <
    array.length; i++)
        current =
        combine(current,
        array[i]);
        return current;
    }
```

```
console.log(reduce([1, 2, 3, 4],
function(a, b) {
        return a + b;
}, 0));
```

REDUCE

- Quem nasceu primeiro?

```
console.log(ancestry.reduce(fun
ction(min, cur) {
        if (_____) return
        cur;

        else return min;

}));
```

REDUCE

- Quem nasceu primeiro?

```
console.log(ancestry.reduce(fun
ction(min, cur) {
        if (cur.born < min.born)
        return cur;
        else return min;
}));
```

SEM REDUCE

```
var min = ancestry[0];
for (var i = 1; i < ancestry.length;
i++) {
        var cur = ancestry[i];
        if (cur.born < min.born)
            min = cur;
}
console.log(min);
```

Composição

```
function average(array) {
    function plus(a, b) { return a
    + b; }

    return array.reduce(plus) /
    array.length;

}
function age(p) { return p.died -
p.born; }
function male(p) { return p.sex
== "m"; }
function female(p) { return p.sex
== "f"; }
```

Composição

```
console.log(average(ancestry.filter(male).map(age)));
console.log(average(ancestry.filter(female).map(age)));
```

- Um pouco de trabalho para construir, mas fica como referência.

Saiba mais

- Eloquent Javascript – Cap. 05
- http://eloquentjavascript.net/05_higher_order.html