JavaScript

Funções

Funções

Declarações

Uma função é definida pela palavra ***function*** seguida de:

1 => Nome da função

2 => Lista de parâmetros da função entre parênteses, separados por vírgulas

3 => Bloco de código javascript delimitado por um par de chaves

```
function square(number) {
  return number * number;
}
```

Funções

Mutabilidade dos parâmetros

```
function myFunc(theObject) {
  theObject.make = "Toyota";
}

const mycar = {
  make: "Honda",
  model: "Accord",
  year: 1998,
};

console.log(mycar.make); // "Honda"
myFunc(mycar);
console.log(mycar.make); // "Toyota"
```

```
function myFunc(theArr) {
  theArr[0] = 30;
}

const arr = [45];

console.log(arr[0]); // 45
myFunc(arr);
console.log(arr[0]); // 30
```

Funções

Expressões

```
const square = function (number) {
  return number * number;
};

console.log(square(4)); // 16
```

```
const factorial = function fac(n) {
  return n < 2 ? 1 : n * fac(n - 1);
};

console.log(factorial(3)); // 6
```

Funções

Expressões

```
function map(f, a) {
  const result = new Array(a.length);
  for (let i = 0; i < a.length; i++) {
    result[i] = f(a[i]);
  }
  return result;
}

const cube = function (x) {
  return x * x * x;
};

const numbers = [0, 1, 2, 5, 10];
console.log(map(cube, numbers)); // [0, 1, 8, 125, 1000]
```

Funções

Recursividade

```
function factorial(n) {
  if (n === 0 || n === 1) {
    return 1;
  } else {
    return n * factorial(n - 1);
  }
}
```

Funções

Function Hoisting

```
console.log(square(5)); // 25

function square(n) {
  return n * n;
}
```

```
console.log(square(5)); // ReferenceError: Cannot access 'square' before initialization
const square = function (n) {
  return n * n;
};
```

Funções

Nested Functions

```
function addSquares(a, b) {
  function square(x) {
    return x * x;
  }
  return square(a) + square(b);
}

console.log(addSquares(2, 3)); // 13
console.log(addSquares(3, 4)); // 25
console.log(addSquares(4, 5)); // 41
```

Funções

Nested Functions

```
function outside(x) {
  function inside(y) {
    return x + y;
  }
  return inside;
}

const fnInside = outside(3); // Think of it like: give me a function that adds 3 to
whatever you give it
console.log(fnInside(5)); // 8
console.log(outside(3)(5)); // 8
```

Funções

Closures

```
// The outer function defines a variable called "name"
const pet = function (name) {
  const getName = function () {
    // The inner function has access to the "name" variable of the outer function
    return name;
  };
  return getName; // Return the inner function, thereby exposing it to outer scopes
};
const myPet = pet("Vivie");

console.log(myPet()); // "Vivie"
```

Funções

Closures

```
const createPet = function (name) {
  let sex;

  const pet = {
    // setName(newName) is equivalent to setName: function (newName)
    // in this context
    setName(newName) {
      name = newName;
    },

    getName() {
      return name;
    },

    getSex() {
      return sex;
    },
    setSex(newSex) {
      if (
        typeof newSex === "string" &&
        (newSex.toLowerCase() === "male" || newSex.toLowerCase() === "female")
      ) {
        sex = newSex;
      }
    },
  };

  return pet;
};

const pet = createPet("Vivie");
console.log(pet.getName()); // Vivie

pet.setName("Oliver");
pet.setSex("male");
console.log(pet.getSex()); // male
console.log(pet.getName()); // Oliver
```

Funções

Objeto Argumentos

```
function myConcat(separator) {
  let result = ""; // initialize list
  // iterate through arguments
  for (let i = 1; i < arguments.length; i++) {
    result += arguments[i] + separator;
  }
  return result;
}
```

Funções

Parâmetros padrão

```
function multiply(a, b) {
  b = typeof b !== "undefined" ? b : 1;
  return a * b;
}

console.log(multiply(5)); // 5
```

```
function multiply(a, b = 1) {
  return a * b;
}

console.log(multiply(5)); // 5
```

Funções

Parâmetros rest

```
function multiply(multiplier, ...theArgs) {
  return theArgs.map((x) => multiplier * x);
}

const arr = multiply(2, 1, 2, 3);
console.log(arr); // [2, 4, 6]
```