Resoluções - Exercícios - Curso Fundamentos de Programação

- Crie uma função que retorna a *string* "Olá, " concatenada com um argumento *text* (a ser passado para a função) e com ponto de exclamação "!" no final.

  Exemplos:

  ```
  cumprimentar("Leonardo") // retornará "Olá, Leonardo!"
  cumprimentar("Maria") // retornará "Olá, Maria!"
  ```

  Resolução 1:

  ```
  function cumprimentar(nome) {
    const saudacao = "Olá"
    return [saudacao, nome].join(', ').concat("!")
  }
  ```

  Resolução 2:

  ```
  function cumprimentar(nome) {
    return "Olá, " + nome + "!"
  }
  ```

  Resolução 3:

  ```
  function cumprimentar(nome) {
    return `Olá, ${nome}!`
  }
  ```

- Escreva uma função que receba a idade de uma pessoa em anos e retorne a mesma idade em dias.

  > Obs: considere que um ano tem 365 dias. Desconsidere anos bissextos (com 366 dias) e desconsidere também dias decorridos desde o último aniversário.

  Exemplos:

  ```
  converterIdadeEmAnosParaDias(25) // retornará 9125
  converterIdadeEmAnosParaDias(70) // retornará 25550
  ```

  Resolução:

  ```
  function converterIdadeEmAnosParaDias(idadeEmAnos) {
    const diasDoAno = 365

    return diasDoAno * idadeEmAnos
  }
  ```

- Desenvolva uma função que recebe dois parâmetros, um é a quantidade de horas trabalhadas por um funcionário num mês, e o quanto ele recebe por hora. A função deverá calcular o salário líquido mensal do funcionário, que é da quantidade de horas trabalhadas no mês multiplicada pelo valor da sua hora. Desse valor, deve ser subtraído 30%, relativo a impostos.

O retorno da função deve ser a string "Salário líquido é igual a R$ X", em que X é o salário líquido do funcionário no mês.

Exemplos:

```
calcularSalarioLiquido(180, 60) // retornará "Salário igual a R$ 7560"
```

Resolução:

```
function calcularSalarioLiquido(horasTrabalhadas, ganhoPorHora) {
  const salarioBruto = horasTrabalhadas * ganhoPorHora
  const salarioLiquido = salarioBruto - salarioBruto * 30/100

  return `Salário igual a R$ ${salarioLiquido}`
}
```

- Crie uma função que recebe um número (de 1 a 12) e retorne o mês correspondente como uma *string*. Por exemplo, se a entrada for 2, a função deverá retornar "fevereiro", pois este é o 2° mês.

  Exemplos:

  ```
  receberNomeDoMes(1) // retornará "janeiro"
  receberNomeDoMes(4) // retornará "abril"
  ```

  Resolução 1:

  ```
  function receberNomeDoMes(numero) {
    switch(numero){
      case 1:
        return "janeiro";
      case 2:
        return "fevereiro";
      case 3:
        return "março";
      case 4:
        return "abril";
      case 5:
        return "maio";
      case 6:
        return "junho";
      case 7:
        return "julho";
      case 8:
        return "agosto";
      case 9:
        return "setembro";
      case 10:
        return "outubro";
      case 11:
        return "novembro";
      case 12:
        return "dezembro";
    }
  }
  ```

  Resolução 2:

  ```
  function receberNomeDoMes(numero) {
    const mapeamento = ['janeiro', 'fevereiro', 'março', 'abril', 'maio', 'junho', 'julho',
      'agosto', 'setembro', 'outubro', 'novembro', 'dezembro']

    return mapeamento[--numero];
  }
  ```

- Crie uma função que receba dois números e retorne se o primeiro é maior ou igual ao segundo. Deverá diferenciar números de *strings*.

  Exemplos:

```
maiorOuIgual(0, 0) // retornará true
maiorOuIgual(0, "0") // retornará false
maiorOuIgual(5, 1) // retornará true
```

Resolução:

```
function maiorOuIgual(primeiro, segundo) {
  if(typeof primeiro != typeof segundo) return false

  return primeiro >= segundo
}
```

- Escreva uma função que receba um valor booleano ou numérico. Se o parâmetro fornecido for booleano, o retorno da função deverá ser o inverso. Por exemplo, se a entrada for *false*, retornará *true*. Se o parâmetro for numérico, o retorno será o número inverso. Por exemplo, se for fornecido 1, o retorno será -1. Se o parâmetro de entrada não for de nenhum dos tipo acima, retorne "booleano ou número esperados, mas o parâmetro é do tipo ...".

  Exemplos:

  ```
  inverso(true) // retornará false
  inverso("6") // retornará "booleano ou número esperados, mas o parâmetro é do tipo string"
  inverso(-2000) // retornará 2000
  inverso("programação") // retornará "booleano ou números, mas o parâmetro é do tipo string"
  ```

  Resolução:

  ```
  function inverso(valor) {
    const tipo = typeof valor

    if (tipo  == "boolean") return !valor
    else if (tipo  == "number") return -valor
    else
      return `booleano ou número esperados, mas o parâmetro é do tipo ${tipo}`
  }
  ```

- Crie uma função que receba quatro números como parâmetro (*numero, minimo*, *maximo, inclusivo*) e retorne se o parâmetro *numero* (o primeiro) está entre *minimo* e *maximo*. *Q*uando o parâmetro *inclusivo* for *true,* tenha "entre" como inlusivo, ou seja, considerando se *numero* é igual a *minimo* ou a *maximo*. Caso o parâmetro *inclusivo* não seja informado, seu valor padrão deverá ser *false,* portanto, a lógica será exclusiva, não considerando se *numero* é igual a *minimo* ou a *maximo*.

  Exemplos:

  ```
  estaEntre(10, 100, 50) // retornará true
  estaEntre(16, 100, 160) // retornará false
  estaEntre(3, 150, 3) // retornará false
  estaEntre(3, 150, 3, true) // retornará true
  ```

  Resolução 1:

  ```
  function estaEntre(minimo, maximo, numero, inclusivo = false) {
    if(inclusivo) return numero >= minimo && numero <= maximo

    return numero > minimo && numero < maximo
  }
  ```

  Resolução 2:

  ```
  function estaEntre(minimo, maximo, numero, inclusivo = false) {
    return inclusivo ? numero >= minimo && numero <= maximo : numero > minimo && numero < maximo
  }
  ```

- Desenvolva uma função que recebe dois números inteiros não negativos (maiores ou iguais a zero) e realize a multiplicação deles. Porém, não utilize o operador de mutiplicação.

  Exemplo:

```
multiplicar(5, 5) // retornará 25
multiplicar(0, 7) // retornará 0
```

  Resolução 1:

```
function multiplicar(numeroA, numeroB) {
  let resultado = 0

  /* a otimização feita para diminuir a quantidade de chamadas recursivas pode ser
     realizada aqui para diminuir a quantidade de loops */
  for(let i = 0; i < numeroB; i++)
    resultado += numeroA

  return resultado
}
```

  Resolução 2:

```
function multiplicar(numero, multiplicador) {
  if (numero === 0 || multiplicador === 0) return 0

  return multiplicador === 1 ? numero : numero + multiplicar(numero, multiplicador-1)
}
```

  Resolução 3:

```
function multiplicar(numeroA, numeroB) {
  if (numeroA === 0 || numeroB === 0) return 0

  const maiorNumero = Math.max(numeroA, numeroB)
  const menorNumero = Math.min(numeroA, numeroB)

  function multiplicarRecursivamente(numero, multiplicador) {
    return (
      multiplicador === 1 ?
      numero :
      numero + multiplicarRecursivamente(numero, multiplicador-1)
    )
  }

  // nessa versão, garantimos que o multiplicador será será o menor número, portanto haverá o mínimo de chamadas recursivas
  return multiplicarRecursivamente(maiorNumero, menorNumero )
}
```

- Escreva uma função que receba dois parâmetros. O primeiro parâmetro é o elemento que repetirá, enquanto que o segundo será o número de vezes que haverá repetição. Um array será retornado.

  Exemplos:

```
repetir("código", 2) // retornará ["código", "código"]
repetir(7, 3) // retornará [7, 7, 7]
```

  Resolução 1:

```
function repetir(item, quantidade) {
  let resultado = []

  for (let i = 0; i < quantidade; i++)
    resultado.push(item)

  return resultado
}
```

  Resolução 2:

```
function repetir(item, quantidade) {
  return Array(quantidade).fill(item)
}
```

- Elabore uma função que recebe um número como parâmetro e retorne uma string com o símbolo "+" na quantidade especificada no parâmetro.

  Exemplos:

  ```
  simboloMais(2) // retornará "++"
  simboloMais(4) // retornará "++++"
  ```

  Resolução 1:

  ```
  function simboloMais(quantidade) {
   let resultado = ''

    for(let i = 0; i < quantidade; i++)
      resultado += '+'

    return resultado
  }
  ```

  Resolução 2:

  ```
  function simboloMais(quantidade) {
    return Array(quantidade).fill('+').join('')
  }
  ```

  Resolução 3:

  ```
  function simboloMais(quantidade) {
    return "+".repeat(quantidade)
  }
  ```

- Crie uma função que receba uma *array* e retorne o primeiro e o último elemento desse array como um novo *array*:

  Exemplos:

  ```
  receberPrimeiroEUltimoElemento([7,14,"olá"]) // retornará [7, "olá"]
  receberPrimeiroEUltimoElemento([-100, "aplicativo", 16]) // retornará [-100, 16]
  ```

  Resolução 1:

  ```
  function receberPrimeiroEUltimoElemento(elementos) {
    const indiceDoPrimeiroElemento = 0
    const indiceDoUltimoElemento = elementos.length - 1
    const primeiroElemento = elementos[indiceDoPrimeiroElemento]
    const ultimoElemento = elementos[indiceDoUltimoElemento]

    return [primeiroElemento, ultimoElemento]
  }
  ```

  Resolução 2:

  ```
  function receberPrimeiroEUltimoElemento(elementos) {
    const primeiroElemento= elementos.shift()
    const ultimoElemento = elementos.pop()

    return [primeiroElemento, ultimoElemento]
  }
  ```

  Resolução 3:

```
function receberPrimeiroEUltimoElemento([primeiroElemento, ...elementosRestantes]) {
  const ultimoElemento = elementosRestantes.pop()

  return [primeiroElemento, ultimoElemento]
}
```

- Quando temos um objeto e manipulamos seus atributos, adicionando, atualizando ou removendo-os, estamos apenas modificando-o, mas, em essência, o objeto continua o mesmo, ou seja a sua referência de memória é a mesma.

  Num projeto que você está trabalhando, você foi designado a refatorar diversas funções para que façam cópias de objetos e manipulem os dados dessas cópias, com o intuito de evitar efeitos indesejáveis em algumas situações devido a referências a objetos. Abaixo, está a descrição de uma dessas funções.

  Você escreverá uma função que recebe um objeto como primeiro parâmetro e, como segundo parâmetro, o nome de uma propriedade contida nesse objeto. Em seguida, retorne uma cópia desse objeto sem a propriedade especificada no segundo parâmetro.

  Exemplos:

  ```
  removerPropriedade({a: 1, b: 2}, "a") // retornará {b: 2}
  removerPropriedade({
    id: 20,
    nome: "caneta",
    descricao: "Não preenchido"
  }, "descricao") // retornará {id: 20, nome: "caneta"}
  ```


  A fim de testar se o objeto retornado não é o mesmo que foi passado como parâmetro para a função *removerPropriedade*, você poderá usar a função *Object.is()*, por exemplo:

  `Object.is(removerPropriedade(objeto, "descricao"), objeto)`

  Retornará *false* se o objeto não for o mesmo.

  Resolução 1:

  ```
  function removerPropriedade(objeto, nomeDaPropriedade) {
    const copia = Object.assign({}, objeto)
    delete copia[nomeDaPropriedade]

    return copia
  }
  ```

  Resolução 2:

  ```
  function removerPropriedade(objeto, nomeDaPropriedade) {
    const copia = {...objeto}
    delete copia[nomeDaPropriedade]

    return copia
  }
  ```

- Crie uma função que receba um array de elementos e retorne um array somente com os números presentes no array recebido como parâmetro.

  Exemplos:

  ```
  filtrarNumeros(["Javascript", 1, "3", "Web", 20]) // retornará [1, 20]
  filtrarNumeros(["a", "c"]) // retornará []
  ```

  Resolução 1:

  ```
  function filtrarNumeros(array) {
    const resultado = []
  ```

```
  for(let item of array)
    if(typeof item === "number")
      resultado.push(item)

  return resultado
}
```

Resolução 2:

```
function filtrarNumeros(array) {
  return array.filter(item => typeof item === "number")
}
```

- Desenvolva uma função que recebe como parâmetro um objeto e retorne um array de arrays, em que cada elemento é um array formado pelos pares *chave/valor* que corresponde a um atributo do objeto. Observe os exemplos abaixo para um melhor entendimento:

  Exemplos:

```
objetoParaArray({
  nome: "Maria",
  profissao: "Desenvolvedora de software"
}) // irá retornar [["nome", "Maria"], ["profissao", "Desenvolvedora de Software"]]

objetoParaArray({
  codigo: 11111,
  preco: 12000
}) // irá retornar [["codigo", 11111], ["preco", 12000]]
```

  Resolução 1:

```
function objetoParaArray(objeto) {
  const resultado = []

  for (let chave in objeto)
    resultado.push([ chave , objeto[chave] ])

  return resultado
}
```

  Resolução 2:

```
function objetoParaArray(objeto) {
  const chaves = Object.keys(objeto)
  const resultado = chaves.map( chave => [chave, objeto[chave]] )

  return resultado
}
```

  Resolução 3:

```
function objetoParaArray(objeto) {
  return Object.entries(objeto)
}
```

- Elabore uma função que receba um *array* de números e retorne um array que tenha todos os números que são pares e que também tenham índices pares.

  Lembre-se que um número é par porque é divisível por 2, ou seja, o resto da divisão da divisão dele por 2 é zero.

  Exemplos:

```
receberSomenteOsParesDeIndicesPares([1, 2, 3, 4]) // retornará []
receberSomenteOsParesDeIndicesPares([10, 70, 22, 43]) // retornará [10, 22]
```

Resolução 1:

```
function receberSomenteOsParesDeIndicesPares(numeros) {
  let resultado = []

  for(let i = 0; i < numeros.length; i += 2){
    const numeroPar = numeros[i] % 2 === 0

    if(numeroPar)
      resultado.push(numeros[i])
  }

  return resultado
}
```

Resolução 2:

```
function receberSomenteOsParesDeIndicesPares(numeros) {
  return numeros.filter((numero, indice) => {
    const numeroPar = numero % 2 === 0
    const indicePar = indice % 2 === 0

    return numeroPar && indicePar
  })
}
```

- A fim de manter o calendário anual ajustado com o movimento de translação da Terra, criou-se os anos bissextos, que têm 366 dias em vez dos 365 presentes nos anos normais.

  Para determinar se um ano é bissexto, é necessário saber se ele é multiplo de 4. Não pode ser múltiplo de 100, exceto se for também múltiplo de 400.

  Com isso em mente, desenvolva uma função que recebe um número correspondente a um ano e retorna se ele é bissexto ou não.

  Exemplos:

  ```
  checarAnoBissexto(2020) // retornará true
  checarAnoBissexto(2100) // retornará false, pois é múltiplo de 100 e não é múltiplo de 400
  ```

  Resolução 1:

  ```
  function checarAnoBissexto(ano) {
    const divisivelPorQuatro = ano % 4 == 0
    const divisivelPorCem = ano % 100 == 0
    const divisivelPorQuatrocentos = ano % 400 == 0

    return (divisivelPorQuatro && !divisivelPorCem) || divisivelPorQuatrocentos
  }
  ```

  Resolução 2:

  ```
  // checa-se indiretamente, verificando se o mês de fevereiro do dado ano tem 29 dias
  function checarAnoBissexto(ano) {
    return new Date(ano, 1, 29).getDate() === 29;
  }
  ```

- Escreva uma função que receba um array de números e retornará a soma de todos os números desse array.

  Exemplos:

  ```
  somarNumeros([10, 10, 10]) // retornará 30
  somarNumeros([15, 15, 15, 15]) // retornará 60
  ```

  Resolução 1:

```
function somarNumeros(numeros) {
  const quantidadeDeNumeros = numeros.length

  return (quantidadeDeNumeros === 0) ? 0 : numeros[0] + somarNumeros(numeros.slice(1))
}
```

Resolução 2:

```
function somarNumeros(numeros) {
  let soma = 0
  numeros.forEach(numero => soma += numero)

  return soma
}
```

Resolução 3:

```
function somarNumeros(numeros) {
  const soma = numeros.reduce( (acumulador, numeroAtual) => acumulador + numeroAtual, 0)

  return soma
}
```

- Você está trabalhando numa aplicação pessoal de controle de despesas. Na tela principal dessa aplicação, é possível adicionar produtos ou serviços, informando nome, categoria e preço. Uma funcionalidade que você está desenvolvendo no momento é a de somar o total das despesas.

  Crie uma função que receba um *array* de produtos e retorne o total das despesas.

  Exemplos:

```
despesasTotais([
  {nome: "Jornal online", categoria: "Informação", preco: 89.99},
  {nome: "Cinema", categoria: "Entretenimento", preco: 150}
]) // retornará 239.99

despesasTotais([
  {nome: "Galaxy S20", categoria: "Eletrônicos", preco: 3599.99},
  {nome: "Macbook Pro", categoria: "Eletrônicos", preco: 30999.90}
]) // retornará 34599.89
```

  Resolução 1:

```
function despesasTotais(itens) {
  var total = 0

  for (let item of itens)
    total += item.preco

  return total
}
```

  Resolução 2:

```
function despesasTotais(itens) {
  return itens
    .map(item => item.preco)
    .reduce((acumulador, valorAtual) => acumulador + valorAtual)
}
```

  Resolução 3:

```
function despesasTotais(itens) {
  return itens.reduce((acumulador, valorAtual) => acumulador + valorAtual.preco, 0)
}
```

- Numa aplicação Web de investimento financeiro da qual você faz parte da equipe de desenvolvimento, pretende-se adicionar a funcionalidade de calcular a média de um conjunto de números informados pelo usuário.

  Com o intuito de realizar esse cálculo, crie uma função que receba um array com uma quantidade indeterminada de números e retorne a média simples desses números.

   A média simples é o resultado da soma de todos os números dividido pela quantidade de números.

  Exemplos:

```
calcularMedia([0, 10]) // retornará 5
calcularMedia([1, 2, 3, 4, 5]) // retornará 3
```

  Resolução 1:

```
function calcularMedia(numeros) {
  const quantidadeDeNumeros = numeros.length
  let somaTotal = 0

  for(numero of numeros) {
    somaTotal += numero
  }

  return somaTotal / quantidadeDeNumeros
}
```

  Resolução 2:

```
function calcularMedia(numeros) {
  const quantidadeDeNumeros = numeros.length
  const somaTotal = numeros.reduce((numeroA, numeroB) => numeroA + numeroB)

  return somaTotal / quantidadeDeNumeros
}
```

16 - Faça uma função que recebe a base e a altura de um triângulo e retorne a área desse triângulo. A precisão deverá ser de duas casas decimais, arredondando se necessário.

   Obs: a fórmula para calcular a área de um triângulo é *(base x altura) / 2*

  Exemplos:

```
areaDoTriangulo(10, 15) // retornará "75,00"
areaDoTriangulo(7, 9) // retornará "31,50"
areaDoTriangulo(9.25, 13.1) // retornará "60.59"
```

  Resolução:

```
function areaDoTriangulo(base, altura) {
  const area = (base * altura) / 2

  return area.toFixed(2) // irá arredondar para manter 2 casas decimais
}
```

17 - Criar uma função que receba um array de números e retorne o menor número desse array.

  Exemplos:

```
menorNumero([10, 5, 35, 65]) // retornará 5
menorNumero([5, -15, 50, 3]) // retornará -15
```

Resolução 1:

```
function menorNumero(numeros) {
  let menor = numeros[0]

  for (let i in numeros)
    if (numeros[i] < menor)
      menor = numeros[i]

  return menor
}
```

Resolução 2:

```
function menorNumero(numeros) {
  return numeros.reduce((anterior, atual) => anterior < atual ? anterior : atual)
}
```

Resolução 3:

```
function menorNumero(numeros) {
  return Math.min(...numeros);
}
```

- Desenvolva uma função que receba como parâmetro um número de 1 a 10. Internamente, na função, será gerado um número aleatório de 1 a 10. A função deverá retornar se o parâmetro de entrada foi igual ao número sorteado internamente. Se o valor fornecido foi o sorteado, retorne "Parabéns! O número sorteado foi o X". Se não for igual, retorne "Que pena! O número sorteado foi o X". X é o número que foi sorteado.

  Exemplos:

  ```
  funcaoDaSorte(10) // retornará "Parabéns! O número sorteado foi o 10"
  funcaoDaSorte(5) // retornará "Que pena! O número sorteado foi o 3"
  funcaoDaSorte(5) // retornará "Que pena! O número sorteado foi o 1"
  ```

  Resolução:

  ```
  /*mais informações sobre gerar números aleatórios dentro de uma certa faixa
    https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/random */
  function funcaoDaSorte(numeroEscolhido) {
    const min = 1
    const max = 10
    const numeroAleatorio = Math.floor(Math.random() * (max - min + 1) + min)

    return (numeroEscolhido === numeroAleatorio ?
      `Parabéns! O número sorteado foi o ${numeroAleatorio}` :
      `Que pena, o número sorteado foi o ${numeroAleatorio}`
    )
  }
  ```

- Criar uma função que receba uma string como parâmetro e conte quantas palavras tem nela.

   Considere que todas as palavras só são separadas por um espaço.

  Exemplos:

  ```
  contarPalavras("Sou uma frase") // retornará 3
  contarPalavras("Me divirto aprendendo a programar") // retornará 5
  ```

  Resolução:

  ```
  function contarPalavras(frase){
    const palavras = frase.split(" ")
  ```

```
  return palavras.length
}
```

- Desenvolva uma função que recebe um caractere e uma string como parâmetros e retorne a quantidade de vezes que o caractere se repete na string. A função deverá ser *case-sensitive*, ou seja, irá diferenciar maiúsculas de minúsculas.

  Exemplos:

```
contarCaractere("r", "A sorte favorece os audazes") // retornará 2
contarCaractere("c", "Conhece-te a ti mesmo") // retornará 1
```

  Resolução 1:

```
function contarCaractere(caractereBuscado, frase) {
  let contador = 0

  for (let i = 0; i < frase.length; i++)
    if (frase.charAt(i) === caractereBuscado)
      contador++

  return contador
}
```

  Resolução 2:

```
function contarCaractere(caractereBuscado, frase) {
  return [...frase].filter(caractere => caractere === caractereBuscado).length
}
```

- A fim de criar um mecanismo de busca para sua aplicação, você precisa iniciar criando uma função para identificar palavras semelhantes.

  Escreva uma função que recebe como primeiro parâmetro uma palavra e, como segundo parâmetro, um *array* de *strings*. A função deverá filtrar as palavras do *array* que contêm nelas a *string* do primeiro parâmetro.

  Exemplos:

```
buscarPalavrasSemelhantes("pro", ["programação", "mobile", "profissional"]) // retornará ["programação", "profissional"]
buscarPalavrasSemelhantes("java", ["javascript", "java", "c++"]) // retornará ["javascript", "java"]
```

  Resolução 1:

```
function buscarPalavrasSemelhantes(inicio, palavras) {
  const resultado = []

  for (let palavra of palavras)
    if (palavra.includes(inicio))
      resultado.push(palavra)

  return resultado
}
```

  Resolução 2:

```
function buscarPalavrasSemelhantes(inicio, palavras) {
  return palavras.filter(palavra => palavra.includes(inicio))
}
```

- Desenvolva uma função que receba uma frase como parâmetro e retorne essa *string* somente com as consoantes, ou seja, sem as vogais.

  Exemplos:

```
removerVogais("Cod3r") // retornará "Cd3r"
removerVogais("Fundamentos") // retornará "Fndmnts"
```

Resolução 1:

```
function removerVogais(frase) {
  const vogais = ["a", "A", "e", "E", "i", "I", "o", "O", "u", "U"]

  vogais.forEach( vogal => frase = frase.replace(vogal, '') )

  return frase
}
```

Resolução 2:

```
function removerVogais(frase) {
  return frase.replace(/[aeiou]/ig, '')
}
```

- Desenvolva uma função que recebe um objeto como parâmetro e retorne um outro objeto que corresponde ao ao objeto recebido como parâmetro, porém com as posições das chaves e valores invertidas, conforme exemplo a seguir:

  Exemplo:

  ```
  inverter({ a: 1, b: 2, c: 3}) // retornará { 1: "a", 2: "b", 3: "c"}
  ```

  Resolução 1:

  ```
  function inverter(objeto) {
    const objetoInvertido = {}

    Object.entries(objeto).forEach( parChaveValor => {
      const chave = 0,
            valor = 1

      objetoInvertido[ parChaveValor[valor] ] = parChaveValor[chave]
    })

    return objetoInvertido
  }
  ```

  Resolução 2:

  ```
  function inverter(objeto) {
    const paresDeChaveValorInvertidos = Object.entries(objeto)
            .map( parChaveValor => parChaveValor.reverse() )

    return Object.fromEntries(paresDeChaveValorInvertidos)
  }
  ```

- Elabore uma função que recebe dois parâmetros: o primeiro é um array de números e o segundo é um número que especifica uma quantidade de dígitos. Essa função deverá retornar somente aqueles números do array que têm a quantidade de dígitos indicada pelo segundo parâmetro.

  Exemplos:

  ```
  filtrarPorQuantidadeDeDigitos([38, 2, 365, 10, 125, 11], 2) // retornará [38, 10, 11]
  filtrarPorQuantidadeDeDigitos([5, 9, 1, 125, 11], 1) // retornará [5, 9, 1]
  ```

  Resolução 1:

  ```
  function filtrarPorQuantidadeDeDigitos(numeros, quantidadeDesejada) {
    let resultado = []
  ```

```
  for(numero of numeros){
    const quantidadeDeDigitos = String(numero).length

    if(quantidadeDeDigitos === quantidadeDesejada)
      resultado.push(numero)
  }

  return resultado
}
```

Resolução 2:

```
function filtrarPorQuantidadeDeDigitos(numeros, quantidadeDesejada) {
  return numeros.filter(numero => {
    const quantidadeDeDigitos = String(numero).length

    return quantidadeDeDigitos === quantidadeDesejada
  })
}
```

- Crie uma função que recebe um array de números e retorna o segundo maior número presente nesse array.

  Exemplos:

```
segundoMaior([12, 16, 1, 5]) // retornará 12
segundoMaior([8, 4, 5, 6]) // retornará 6
```

Resolução 1:

```
function segundoMaior(numeros) {
  let indiceDoMaior = 0
  let segundoMaior

  numeros.forEach( (numero, indice) => {
    if( numero > numeros[indiceDoMaior] )
      indiceDoMaior = indice
  })

  numeros.splice(indiceDoMaior, 1)
  segundoMaior = numeros[0]

  numeros.forEach(numero => {
    if(numero > segundoMaior)
      segundoMaior = numero
  })

  return segundoMaior
}
```

Resolução 2:

```
function segundoMaior(numeros) {
  const maiorNumero = Math.max(...numeros)
  numeros = numeros.filter(numero => numero != maiorNumero)
  const segundoMaior = Math.max(...numeros)

  return segundoMaior
}
```

Resolução 3:

```
function segundoMaior(numeros) {
  const numerosOrdenados = numeros.sort((numeroA, numeroB) => numeroB - numeroA)
  const segundoMaior = numerosOrdenados[1]

  return segundoMaior
}
```

- Elabore uma função que recebe um objeto com estudantes e suas notas. As notas de cada estudante estarão num array, conforme exemplo abaixo. Você deverá calcular a média da nota de cada aluno e retornar um objeto com

os atributos *nome* e *media*, que indica o aluno que teve o melhor desempenho nas notas.

Exemplo:

```
recerberMelhorEstudante({
  Joao: [8, 7.6, 8.9, 6], // média 7.625
  Mariana: [9, 6.6, 7.9, 8], // média 7.875
  Carla: [7, 7, 8, 9] // média 7.75
}) // retornará { nome: "Mariana", media: 7.875 }
```

Resolução:

```
const soma = array => array.reduce((acumulador, atual) => acumulador + atual, 0)
const media = array => soma(array) / array.length

function recerberMelhorEstudante(estudantes) {
  const estudantesComMedias = Object.entries(estudantes).map( estudante => {
    const chave = 0,
          valor = 1

    return { nome: estudante[chave], media: media(estudante[valor]) }
  })
  const estudantesOrdenados = estudantesComMedias.sort( (estudanteA, estudanteB) => estudanteB.media - estudanteA.media )
  const melhorEstudante = estudantesOrdenados[0]

  return melhorEstudante
}
```