tryber / **sd-07-project-playground-functions**

generated from betrybe/sd-0x-project-playground-functions

README.md

Boas vindas ao repositório do Projeto Playground Functions!

Você já usa o GitHub diariamente para desenvolver os exercícios, certo? Agora, para desenvolver os projetos, você deverá seguir as instruções a seguir. Fique atento a cada passo, e se tiver qualquer dúvida, nos envie por *Slack*! #VQV 🚀

Aqui você vai encontrar os detalhes de como estruturar o desenvolvimento do seu projeto a partir desse repositório, utilizando uma `branch` específica e um *Pull Request* para colocar seus códigos.

📝 ANTES DE COMEÇAR A DESENVOLVER:

Tutorial em Vídeo

1. Clone o repositório

- `git clone https://github.com/tryber/sd-07-project-playground-functions.git .`
- Entre na pasta do repositório que você acabou de clonar:
  - `cd sd-07-project-playground-functions`

2. Crie uma branch a partir da branch `master`

- Verifique que você está na branch `master`
  - Exemplo: `git branch`
- Se não estiver, mude para a branch `master`
  - Exemplo: `git checkout master`
- Agora, crie uma branch onde você vai guardar os `commits` do seu projeto
  - Você deve criar uma branch no seguinte formato: `nome-de-usuario-nome-do-projeto`
  - Exemplo: `git checkout -b joaozinho-project-playground-functions`

3. Adicione as mudanças ao *stage* do Git e faça um `commit`

- Verifique que as mudanças ainda não estão no *stage*
  - Exemplo: `git status` (devem aparecer listados os novos arquivos em vermelho)
- Adicione o novo arquivo ao *stage* do Git
  - Exemplo:
    - `git add .` (adicionando todas as mudanças - *que estavam em vermelho* - ao stage do Git)
    - `git status` (devem aparecer listados os arquivos em verde)
- Faça o `commit` inicial
  - Exemplo:
    - `git commit -m 'iniciando o projeto. VAMOS COM TUDO :rocket:'` (fazendo o primeiro commit)
    - `git status` (deve aparecer uma mensagem tipo *nothing to commit*)

4. Adicione a sua branch com o novo `commit` ao repositório remoto

- Usando o exemplo anterior: `git push -u origin joaozinho-project-playground-functions`

5. Crie um novo `Pull Request` *(PR)*

- Vá até a página de *Pull Requests* do repositório no GitHub
- Clique no botão verde *"New pull request"*
- Clique na caixa de seleção *"Compare"* e escolha a sua branch **com atenção**
- Clique no botão verde *"Create pull request"*

- Adicione uma descrição para o *Pull Request*, um título claro que o identifique, e clique no botão verde *"Create pull request"*
- **Não se preocupe em preencher mais nada por enquanto!**
- Volte até a página de *Pull Requests* do repositório e confira que o seu *Pull Request* está criado

Entregáveis

Para entregar o seu projeto você deverá criar um *Pull Request* neste repositório.

Este *Pull Request* deverá conter o arquivo `challenges.js` com suas funções implementadas.

Todas as funções já estão declaradas no arquivo `challenges.js`. Você pode criar outras funções para auxiliarem as já existentes. Contudo **Não altere o nome das funções que já existem**.

Os parâmetros das funções já existentes podem e devem ser alterados.

Prazo para entrega

O Prazo para entrega é de **7 dias corridos** após o último dia de projeto.

Exemplo: Se o último dia de projeto aconteceu na **quarta-feira, dia 17 de junho**, seu prazo final de entrega será na **quarta-feira 24 de junho** às **14 horas**.

Vale ressaltar que os projetos podem ter mais de um dia de duração, por isso o prazo de **7 dias** é contado à partir do último dia de projeto.

⚠️ É importante que seus arquivos não tenham o nome alterado! ⚠️

Lembre-se que você pode consultar nosso conteúdo sobre Git & GitHub sempre que precisar!

Requisitos do projeto

⚠️ Leia-os atentamente e siga à risca o que for pedido. Não altere o nome de nenhuma função. ⚠️

O não cumprimento de um requisito, total ou parcialmente, impactará em sua avaliação.

👀 Observações importantes:

- Para verificar se a sua função foi criada corretamente você pode instalar a extensão `code runner` no *VSCode*.

1 - Usando o operador &&

JavaScript possui um operador lógico `&&` , o qual recebe dois valores e retorna `true` se ambos os valores são verdadeiros, e retorna `false` se algum dos valores não o for.

Considerando isso, crie uma função chamada `compareTrue` que, ao receber dois booleanos:

- Retorne `true` se ambos os valores são verdadeiros;
- Retorne `false` se um ou ambos os parâmetros forem falsos.

Faça a função utilizando o operador `&&` .

2 - Área do triângulo

Escreva uma função com o nome `calcArea` que receba um valor de base (chamado `base`) e outro de altura (chamado `height`) de um triângulo e retorne o cálculo da sua área.

Lembre-se que a área de um triângulo é calculada através da seguinte fórmula: (base * altura) / 2.

3 - Dividindo a frase

Escreva uma função com o nome `splitSentence` , a qual receberá uma string e retornará uma array de strings separadas por cada espaço na string original.

Exemplo: se a função receber a string `"go Trybe"` , o retorno deverá ser `['go', 'Trybe']` .

4 - Concatenação de strings

Escreva uma função com o nome `concatName` que, ao receber uma array de strings, retorne uma string com o formato `'ÚLTIMO ITEM, PRIMEIRO ITEM'` , independente do tamanho da array.

Isso quer dizer que, caso o parâmetro passado para `concatName` seja a Array ['Lucas', 'Cassiano', 'Ferraz', 'Paolillo'], a função deverá retornar `Paolillo, Lucas` .

5 - Pontos no futebol

Escreva uma função com o nome `footballPoints` que receba o número de vitórias (esse parâmetro deverá se chamar `wins`) e o número de empates (esse parâmetro deverá se chamar `ties`) e retorne a quantidade de pontos que o time marcou em um campeonato.

Para tanto, considere que cada vitória vale 3 pontos e cada empate vale 1 ponto.

6 - Repetição do maior número

Escreva uma função chamada `highestCount` que, ao receber uma array de números, retorne a quantidade de vezes que o maior deles se repete.

Exemplo: caso o parâmetro de `highestCount` seja uma array com valores `[9, 1, 2, 3, 9, 5, 7]` , a função deverá retornar `2` , que é a quantidade de vezes que o número `9` (maior número do array) se repete.

7 - Caça ao rato

Imagine que existem dois gatos, os quais chamaremos de `cat1` e `cat2` , e que ambos estão atrás de um rato chamado `mouse` . Imagine que cada um dos três animais está em uma posição representada por um número.

Sabendo disso, crie uma função chamada `catAndMouse` que, ao receber a posição de `mouse` , `cat1` e `cat2` , **nessa ordem**, calcule as distâncias entre o rato e os gatos e retorne qual dos felinos irá alcançar o rato primeiro (sendo aquele que estará mais perto).

Exemplo: caso o gato `cat2` esteja a 2 unidades de distância do rato, e `cat1` esteja a 3 unidades, sua função deverá retornar `cat2` .

Caso os gatos estejam na mesma distância do rato, a função deverá retornar a string `"os gatos trombam e o rato foge"` .

8 - FizzBuzz

Crie uma função chamada `fizzBuzz` que receba uma array de números e retorne uma array da seguinte forma:

- Para cada número da Array que seja divisível apenas por 3, apresente uma string `"fizz"`;
- Para cada número da Array que seja divisível apenas por 5, apresente uma string `"buzz"`;
- Caso o número seja divisível por 3 e 5, retorne a string `"fizzBuzz"`;
- Caso o número não possa ser dividido por 3 nem por 5, retorne a string `"bug!"`;

Exemplo: caso o parâmetro seja [2, 15, 7, 9, 45], sua função deverá retornar `["bug!", "fizzBuzz", "bug!", "fizz", "fizzBuzz"]`.

9 - Codifique e Decodifique

Crie duas funções: a primeira deverá se chamar `encode` e, ao receber uma string como parâmetro, deverá trocar todas as vogais minúsculas por números, de acordo com o formato a seguir:

a -> 1
e -> 2
i -> 3
o -> 4
u -> 5

Ou seja, caso o parâmetro de `encode` seja `"hi there!"`, o retorno deverá ser `"h3 th2r2!"`.

A segunda função deverá se chamar `decode` e faz o contrário de `encode` - ou seja, recebe uma string contendo números no lugar de letras minúsculas e retornará uma string com vogais minúsculas no lugar dos números (então, caso o parâmetro de `decode` seja `"h3 th2r2!"`, o retorno deverá ser `"hi there!"`).

10 - Lista de tecnologias

Crie uma função que recebe um array de nomes de tecnologias que você quer aprender. Essa função deve receber também um segundo parâmetro chamado `name` com um nome.

Para cada tecnologia no array, crie um objeto com a seguinte estrutura:

```
{
  tech: "NomeTech",
  name: name
}
```

Estes objetos devem ser inseridos em uma nova lista em ordem crescente a partir do campo `tech` no objeto.

A saída da sua função deve ser uma lista de objetos ordenada pelo campo `tech` dos objetos com o formato acima.

Exemplo:

```
Entradas da função:

["React", "Jest", "HTML", "CSS", "JavaScript"]
"Lucas"

// Saída:

[
  {
    tech: "CSS",
    name: "Lucas"
  },
  {
    tech: "HTML",
    name: "Lucas"
  },
  {
    tech: "JavaScript",
    name: "Lucas"
  },
  {
    tech: "Jest",
    name: "Lucas"
  },
  {
    tech: "React",
    name: "Lucas"
  }
]
```

Caso o array venha vazio sua função deve retornar 'Vazio!'

Bônus

11 - Número de telefone

Crie uma função chamada `generatePhoneNumber` que receba uma array com 11 números e retorne um número de telefone, respeitando parênteses, traços e espaços.

Exemplo: caso o parâmetro da função seja [1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1], `generatePhoneNumber` deverá retornar `(12) 34567-8901`.

- Se a função receber um array com tamanho diferente de 11, a mesma deve retornar `"Array com tamanho incorreto."`.

- Caso algum dos números da array seja menor que 0, maior que 9 ou se repita 3 vezes ou mais, `generatePhoneNumber` deverá retornar a string `"não é possível gerar um número de telefone com esses valores"`.

12 - Condição de existência de um triângulo

Um triângulo é composto de três linhas: `lineA`, `lineB` e `lineC`. Crie uma função chamada `triangleCheck` que deverá receber as três linhas como parâmetro e retornar se é possível formar um triângulo com os valores apresentados de cada linha

Para tanto, tenha em mente algumas considerações:

- Para que seja possível formar um triângulo, é necessário que a medida de qualquer um dos lados seja menor que a soma das medidas dos outros dois e maior que o valor absoluto da diferença entre essas medidas.

- Para obter o valor absoluto de um número em JavaScript, pesquise pela função `Math.abs`.

- O retorno da sua função deverá ser um booleano.

Exemplo: o retorno de `triangleCheck(10, 14, 8)` deverá ser `true`.

13 - Bem vindo ao Bar da Trybe!

Segundo as regras desse bar, a cada bebida deve-se beber um copo de água para que não se tenha ressaca.

Crie a função `hydrate` que recebe uma string, e retorne a sugestão de quantos copos de água você deve beber. Exemplos:

```
String recebida:
  "1 cerveja"
String retornada:
  "1 copo de água"
```

```
String recebida:
  "1 cachaça, 5 cervejas e 1 copo de vinho"
String retornada:
  "7 copos de água"
```

```
String recebida:
  "1 cachaça, 5 cervejas e 1 copo de vinho"
String retornada:
  "7 copos de água"
```

Notas

- Para simplificar, consideraremos que qualquer coisa com um número à frente é uma bebida **e que a sua string sempre virá com o formato quantidade (em número) + tipo da bebida**.
- O número na frente de cada bebida está no intervalo entre 1 e 9.

Dica: pesquise por algo similar a `get all integers inside a string js`.

DURANTE O DESENVOLVIMENTO

- Faça `commits` das alterações que você fizer no código regularmente;
- Lembre-se de sempre após um ~~(ou alguns)~~ `commits` atualizar o repositório remoto (o famoso `git push`);
- Os comandos que você utilizará com mais frequência são:
  i. `git status` *(para verificar o que está em vermelho - fora do stage - e o que está em verde - no stage)*;
  ii. `git add` *(para adicionar arquivos ao stage do Git)*;
  iii. `git commit` *(para criar um commit com os arquivos que estão no stage do Git)*;
  iv. `git push -u nome-da-branch` *(para enviar o commit para o repositório remoto na primeira vez que fizer o `push` de uma nova branch)*;
  v. `git push` *(para enviar o commit para o repositório remoto após o passo anterior)*.

DEPOIS DE TERMINAR O DESENVOLVIMENTO - OPCIONAL, PORÉM MUITO IMPORTANTE! <3

Para sinalizar que o seu projeto está pronto para o *"Code Review"* dos seus colegas, faça o seguinte:

- Vá até a página **DO SEU** *Pull Request*, adicione a label de *"code-review"* e marque seus colegas:

- No menu à direita, clique no *link* **"Labels"** e escolha a *label* **code-review**;
- No menu à direita, clique no *link* **"Assignees"** e escolha **o seu usuário**;
- No menu à direita, clique no *link* **"Reviewers"** e digite `students`, selecione o time `tryber/students-sd-07`.

Caso tenha alguma dúvida, aqui tem um video explicativo.

REVISANDO UM PULL REQUEST

Use o conteúdo sobre Code Review para te ajudar a revisar os *Pull Requests*.

#VQV 🚀

Releases

No releases published

Packages

No packages published

Contributors 4

 **gustavoabreucaetano** Gustavo Caetano

 **jeanpsv** Jean Paulo Silva Vasconcelos

 **LucasCFerraz** Lucas Cassiano Ferraz Paolillo

 **isaacbatst** Isaac Batista

Languages

- **JavaScript** 100.0%