Atividade de Participação 2 - Revisão Javascript - Parte 2

> Prof. Luiz Gustavo D. de O. Véras
> Desenvolvimento de Sistemas Web (DSWI6)
> Tecnologia em Análise e Desenvolvimento de Sistemas

Atividade 1. Escreva uma classe `Vec` que represente um vetor no espaço bidimensional (Use qualquer uma das notações apresentadas em aula). Ela recebe parâmetros x e y (números), que deve salvar como propriedades com os mesmos nomes.

Forneça ao protótipo/classe de Vec dois métodos, plus e minus, que recebem outro vetor como parâmetro e retornam um novo vetor que contém a soma ou diferença dos valores x e y dos dois vetores (o vetor atual e o parâmetro).

Adicione uma propriedade getter length ao protótipo que calcula o comprimento do vetor - isto é, a distância do ponto (x, y) à origem (0, 0).

Teste sua classe criando dois objetos com `new` e demonstrando as saídas das operações dos métodos implementados na classe.

> **Obs.** Veja como adicionar um método getter aqui

Atividade 2. O operador == compara objetos pela identidade. Mas às vezes é preferível comparar os valores de suas propriedades reais. Escreva uma função chamada `deepEqual` que recebe dois valores e retorna `true` apenas se eles forem o mesmo valor ou forem objetos com as mesmas propriedades, onde os valores das propriedades são iguais quando comparados com uma chamada recursiva a `deepEqual`.

Para descobrir se os valores devem ser comparados diretamente (use o operador `===` para isso) ou se suas propriedades devem ser comparadas, você pode usar o operador `typeof`.

- Se ele produzir "object" para ambos os valores, você deve fazer uma comparação profunda.
- Você precisa levar em conta uma exceção boba: por causa de um acidente histórico, `typeof null` também produz "object".

> A função `Object.keys` será útil quando você precisar percorrer as propriedades dos objetos para compará-las.

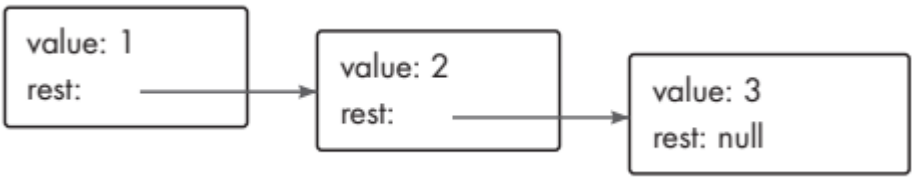
> Os objetos que a função deepEqual são os definidos pela notação `{"param1": "valor1", "param2": "valor"}`.

Considere a explicação a seguir para os exercícios 3, 4, 5 e 6

Objetos, que são os agrupamentos genéricos de valores, podem ser usados para construir diversos tipos de estruturas de dados. Uma estrutura de dados comum é a lista (não confundir com array). Uma lista é um conjunto aninhado de objetos, em que o primeiro objeto mantém uma referência para o segundo, o segundo para o terceiro, e assim por diante.

```
let lista = {
  valor: 1,
  restante: {
    valor: 2,
    restante: {
      valor: 3,
      restante: null
    }
  }
};
```

Os objetos resultantes formam uma cadeia, como segue:



Uma característica interessante das listas é que elas podem compartilhar partes de sua estrutura. Por exemplo, se eu criar dois novos valores `{valor: 0, restante: lista}` e `{valor: -1, restante: lista}` (com `lista` referindo-se à vinculação definida anteriormente), eles são listas independentes, mas compartilham a estrutura que compõe seus últimos três elementos. A lista original ainda é uma lista válida de três elementos.

Atividade 3: Escreva uma função `arrayToList` que construa uma estrutura de lista como a mostrada quando fornecido `[1, 2, 3]` como argumento.

Atividade 4: Também escreva uma função `listToArray` que produza um array a partir de uma lista.

Atividade 5: Adicione uma função auxiliar `prepend`, que recebe um elemento e uma lista, e cria uma nova lista que adiciona o elemento na frente da lista de entrada

Atividade 6: Crie uma funçcao chamada `nth`, que recebe uma lista e um número, e retorna o elemento na posição fornecida na lista (com zero referindo-se ao primeiro elemento) ou `undefined` quando não há tal elemento.

Atividade 7: Utilize o método reduce em combinação com o método concat para "achatar" (*Flattening*) um array de arrays em um único array que contém todos os elementos dos arrays originais.

```
Entrada: [[1, 2, 3], [4, 5], [6]];
Saída: [1, 2, 3, 4, 5, 6];
```

Atividade 8. Faça um script que receba uma data no formato “dd/mm/aaaa” e escreva a data por extenso.

> Dica: use a função “split” de uma string que quebra a string em pedaços dado um separador como argumento da função. Nesse caso, o separador é a barra (/) da data.

> Exemplo: Para a entrada “22/04/1983” deve ser escrito “22 de abril de 1983”.

Atividade 9. Escreva um loop que realiza sete chamadas para console.log para exibir o seguinte triângulo:

```
#
##
###
####
#####
######
#######
```

Pode ser útil saber que você pode encontrar o comprimento de uma string escrevendo .length após ela.

```
let abc = "abc";
console.log(abc.length);
// → 3
```

Atividade 10. Suponha que o método `toUpperCase` não existisse em uma *string*. Implemente uma função **toUpperCase** que retorna o mesmo resultado da original.

Atividade 11: O método `some` da classe array retorna *true* para verificar se pelo menos um dos elementos desse array atende a um critério especificado por uma função passada como argumento. Os arrays possuem também um método chamado `every`. Este retorna *true* quando a função fornecida retorna *true* para cada elemento no array. De certa forma, `some` é uma versão do operador || que atua em arrays, e `every` é como o operador &&.

Implemente a sua própria função `every` como uma função que recebe um array e uma função como parâmetros. Escreva essa versão utilizando um loop.