Casos de teste

Você já viu que é possível utilizar a função pré-definida `console.log()` para exibir no console o resultado da aplicação de funções; dessa maneira, é possível observar se a função está exibindo saídas corretas para cada valor de entrada de dados.

Existe porém, um método sistemático para averiguar se a função está implementada corretamente: o uso de casos de teste. A idéia é definir um conjunto com boa cobertura de diferentes entradas e respectivas saídas corretas e testar o quanto sua função está cobrindo esse conjunto. Se, por exemplo, o conjunto conta com 5 exemplos de teste e sua função funciona corretamente para apenas 3 desses, significa que ela estaria 60 correta e, portanto, merece mais investigação.

Observe o exemplo a seguir para entender o funcionamento dos casos de teste.

[EXEMPLO] Criar uma função que busca o índice de um determinado item em uma lista ORDENADA. Se o item estiver presente, ele deve retornar o índice, caso contrário, deve retornar -1.

```
const {log,indef} = require('./utils.js')

const busca = ([x,...xs],elem,acc=0) => {
  if (indef(x)) return -1
  else if (elem===x) return acc
  else return busca(xs,elem,acc+1)
}

module.exports = {busca}
```

O código acima poderia estar escrito em um arquivo chamado `solucoes.js` , por exemplo.

A ideia então seria criar um arquivo chamado `solucoes.test.js` (pode ter qualquer nomenclatura) onde colocaríamos todos os testes de entrada e saída que desejamos.

```
const {log} = require('./utils.js')
const T = require('./test.js')
const S = require('./solucoes.js')

//CASOS DE TESTE para a função "busca"
T.assert(S.busca([1, 2, 3, 4],3), 2,'busca')
T.assert(S.busca([2, 4, 6, 8, 10], 8), 3,'busca')
T.assert(S.busca([1, 3, 5, 7, 9], 11), -1,'busca')
T.assert(S.busca([1, 5, 7, 11, 25, 100, 200, 350], 5), 1,'busca')

//CASOS DE TESTE para outras funções
...
```

Nas três primeiras linhas fazemos uso de arquivos externos.
O arquivo `utils.js` contém funções genéricas de apoio à implementação.
(para download do arquivo `utils.js`)

O arquivo `test.js` implementa funções para testar aspectos desejados, em particular, a função `assert()` que permite testar cada caso.
(para download do arquivo `test.js`)

O arquivo `solucoes.js` é o arquivo onde a função está implementada, ou seja, o arquivo de solução do problema de fato. Veja que ele precisa ser "enxergado" pelo arquivo de teste, obviamente. E, para isso, sua ultima linha deve conter `module.exports = {busca}` que indica que a função `busca` pode ser "enxergada" por um outro arquivo .js, exatamente como precisamos.

O primeiro parâmetro da função `assert()` é a aplicação da função a ser testada, o segundo parâmetro é o valor correto do resultado e o terceiro parâmetro (opcional) é um texto que o desenvolvedor queira exibir ao lado do resultado do teste; usualmente, o nome da função para facilitar a identificação no console.

Observe a saída da execução do arquivo `solucoes.test.js` acima.

```
busca ✅
busca ✅
busca ✅
busca ✅
```

Caso, por exemplo, o terceiro caso de teste estivesse errado, a saída seria:

busca ✅
busca ✅
busca ❌
busca ✅