

Lista 06



Q1. Sobre o programa a seguir, assinale a alternativa correta.

```
const e = 2.718281828
const f = (x=2) => e**x
console.log(`f: ${f()}`)
```

- a. Função `f` é impura e a aplicação da função está correta.
- b. Função `f` é pura e a aplicação da função está correta.
- c. Função `f` é impura mas a aplicação da função está incorreta.
- d. Função `f` é pura mas a aplicação da função está incorreta.



Q2. Sobre o programa a seguir, assinale a alternativa correta.

```
const a = Math.random()
const g = (num, min, max) => {
  const fator = max - min + 1
  return parseInt(num * fator) + min
}
console.log(`g: ${g(a,1,10)}`)
```

- a. Função `g` é impura porque em sua aplicação, usa-se um valor aleatório como argumento.
- b. Função `g` é impura mas é estável.
- c. Função `g` é impura mas se `const a = Math.Random()` fosse definido dentro da mesma, ela passaria a ser pura.
- d. Função `g` é pura.



Q3. Uma equação de segundo grau genérica se dá por $ax^2 + bx + c$. Observe agora o programa a seguir, que pretende representá-la e viabilizar aplicações a argumentos variados. Assinale a alternativa correta.

```
const h = (a,b,c) => (x) => a*(x**2) + b*x + c
console.log(`h: ${h(-1,2,10)}`)
```

- a. Função `h` representa incorretamente a equação.
- b. Função `h` representa corretamente a equação mas sua aplicação está equivocada.
- c. Função `h` ilustra o que é ser uma Cidadã de Primeira Classe e sua aplicação gera um outro valor.
- d. Função `h` ilustra o que é ser uma Cidadã de Primeira Classe mas sua aplicação está equivocada.



Q4. Observe o programa a seguir e assinale a alternativa correta.

```
const a1 = [1,2,3]
const a2 = a1
a2[0] = -1; a2[1] = -2; a2[2] = -3
console.log(`a1: ${a1}`)
console.log(`a2: ${a2}`)
```

- a. Será impresso a1: 1,2,3 e a2: -1,-2,-3 porque const garante imutabilidade dos elementos das listas.
- b. Será impresso a1: -1,-2,-3 e a2: -1,-2,-3 porque, para listas, const, let e var funcionam da mesma forma.
- c. As listas são mutáveis mas seus elementos, não.
- d. Os elementos da lista a1 são mutáveis, mas a lista a1 é imutável e não seria possível fazer a1 = [-1,-2,-3] logo depois da primeira linha do código.

**Q5. Observe o programa a seguir e assinale a alternativa correta.**

```
const a1 = [1,2,3]
const a2 = a1.map((x)=>x*(-1))
console.log(`a1: ${a1}`)
console.log(`a2: ${a2}`)
```

- a. Será impresso a1: 1,2,3 e a2: -1,-2,-3 porque map garante imutabilidade dos elementos das listas.
- b. Será impresso a1: 1,2,3 e a2: 1,2,3 porque const impediria o efeito da ação do map.
- c. Será impresso a1: 1,2,3 e a2: -1,-2,-3 porque map cria uma nova lista.
- d. O programa possui comportamento igual ao da questão 4.

**Q6. Observe o programa a seguir e assinale a alternativa correta.**


```
const func = () => {
  const nome = 'Cicrana'
  const exibeNome = () => console.log(nome)
  return exibeNome;
}
const minhaFunc = func()
const nome = 'Beltrana'
minhaFunc()
```

- a. O programa exibirá Cicrana como resultado graças ao princípio do Currying
- b. O programa exibirá Cicrana como resultado graças ao princípio do Closure
- c. O programa exibirá Beltrana como resultado graças ao princípio do Currying
- d. O programa exibirá Beltrana como resultado graças ao princípio do Closure

**Q7. Observe o programa a seguir e assinale a alternativa correta.**

```
const exec = (fn) => (...params) => fn(...params)
const soma = (x,y,z) => x+y+z
const multi = (x,y) => x*y
console.log(`Resultado: ${exec(soma)(1,2,3)}`)
console.log(`Resultado: ${exec(multi)(3,5)}`)
```

- a. Uma função `dobro` poderia ser criada facilmente a partir de uma versão currificada de `multi`.
- b. O programa está correto, mas as versões não currificadas das funções `soma` e `multi` melhorariam o reuso.
- c. O programa está incorreto, mas versões currificadas de `soma` e `multi` o tornariam correto.
- d. A possibilidade de se definir `const dobro = multi(2)` demonstra o poder da aplicação parcial neste programa.

 **Q8. Observe o programa abaixo. A função `decrementa` atinge seu objetivo corretamente; entretanto, o programa fere o princípio da Imutabilidade, o que causa problemas com a lista original. Como você corrigiria a inconsistência realizando alterações APENAS nas linhas 1 e 3 do programa?**

```
1 | const l1 = [3, 1, 7]
2 | const decrementa = (lista) => {
3 |     const l2 = lista
4 |     l2[0]--; l2[1]--; l2[2]--;
5 |     return l2
6 | }
7 | console.log(decrementa(l1))
8 | console.log(l1)
```