

Programação recursiva

Vimos anteriormente como conceber uma fórmula recursiva apropriada para variados problemas. Essa formulação recursiva usualmente consiste na identificação de um **Caso Base**, que representa a solução do subproblema mais simples possível e do **Passo Indutivo** (ou **Caso Geral**), que representa a solução de todas as outras situações através da percepção de que um problema maior é composto por problemas menores que compartilham o mesmo algoritmo de solução.

Agora, iremos ver como converter essa formulação recursiva para a implementação da recursividade em Javascript. A abordagem adotada irá considerar a definição de alguns **PADRÕES DE COMPUTAÇÃO RECURSIVA** a fim de treinar o programador no reconhecimento desses padrões em problemas vindouros e, assim, facilitar a concepção das respectivas soluções.

PADRÃO 1: descobrir qual o n-ésimo elemento de uma sequência infinita.



[EXEMPLO] Observe a sequência aritmética a seguir e crie um programa para encontrar o valor do n-ésimo elemento: $\{2, 7, 12, 17, 22, \dots\}$

Formulação recursiva

$$f(1) = 2$$

$$f(n) = f(n - 1) + 5$$

Implementação em Javascript

```
const f = (n) => {
  if (n==1) {return 2}
  else {return f(n-1)+5}
}
```

Versão com *operador condicional ternário*:

```
const f = (n) => (n==0) ? 2 : f(n-1)+5
```



[EXEMPLO: Lista 04, Q2] N-ésimo termo da sequência $\{0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \dots\}$.

Formulação recursiva

$$fib(0) = 0$$

$$fib(1) = 1$$


$$fib(n) = fib(n - 1) + fib(n - 2)$$

Implementação em Javascript

```
const fib = (n) => {
  if (n == 0) {return 0}
  else if (n == 1) {return 1}
  else return fib(n-1) + fib(n-2)
}
```



PADRÃO 2: implementar uma operação que é formada por uma repetição de operações mais primitivas.

 **[EXEMPLO]** Implementar o operador de exponenciação para permitir calcular a potência natural de um número m qualquer: m^n .


Formulação recursiva

$$pot(m, 0) = 1$$

$$pot(m, n) = m * pot(m, n - 1)$$

```
const pot = (m,n) => {
  if (n == 0) {return 1}
  else return m*pot(m,n-1)
}
```



 **[EXEMPLO]** Implementar o operador de exponenciação para permitir calcular a potência *inteira* de um número m qualquer: m^n .

Uso de função de interface

```
const pot = (m,n) => {
  if (n<0) return 1/potAux(m,n*(-1))
  else return potAux(m,n)
}

const potAux = (m,n) => {
  if (n == 0) {return 1}
  else return m*potAux(m,n-1)
}
```



 **[EXEMPLO: Lista 04, Q7]** Implementar o operador de divisão a fim de encontrar o resto da divisão entre dois números inteiros positivos fornecidos, n e m .


Formulação recursiva

$$resto(n, m) = n, \forall n < m$$

$$resto(n, m) = resto(n - m, m), \forall n \geq m$$

```
const resto = (n,m) => n<m ? n : resto(n-m,m)
```



 **[EXEMPLO: Lista 04, Q8]** Implementar o Máximo Divisor Comum (MDC) entre dois inteiros fornecidos, n e m . Naturalmente, você não deve utilizar operadores de divisão da linguagem.

Formulação recursiva

$$mdc(n, m) = m, \text{ se } n = 0$$

$$mdc(n, m) = mdc(n, m - n), \forall m \geq n$$

```
const mdc = (n,m) => {  
  if (n>m) return mdc(m,n)  
  else if (n==0) return m  
  else return mdc(n,m-n)  
}
```

