Mini-demo ⇒ Recursividade

> "Para entender recursividade, você tem que entender recursividade"

Se você já está minimamente familiarizado com funções, deve saber que podemos armazenar o retorno de uma função em uma variável, ou simplesmente utilizá-lo em outras operações. Por exemplo:

```
// Suponha que temos a seguinte função
function returnName(name) {
  return `My first name is ${name}`;
}

console.log(returnName("Wendler")); // My first name is Wendler

// Agora adicionamos mais uma função
function reuturnLastname(lastName) {
  return `my last name is ${lastName}`;
}

console.log(returnName("Tenorio")); // My first name is Tenorio

function returnFullName() {
  return `${returnName("Wendler")}, and ${reuturnLastname("Tenorio")}`;
}


console.log(returnFullName());
```

Foi completamente possível utilizar uma função dentro de outra, nesse caso, chamamos as duas primeiras funções dentro da terceira, e seus valores de retorno foram concatenados em uma única string.

Um outro exemplo:

```
function sum5(number) {
  return number + 5;
```

```
}

function power2(number) {
  return number**2;
}

console.log(sum5(5)); // 5 + 5 => 10
console.log(power2(5)); // 5**2 => 25

// Poderia alterar a segunda função para
function multiplyTo(number) {
  return number * sum5(number);
}

console.log(multiplyTo(5)); // 5 * (5 + 5) => 50
```

Bem útil não? Se podemos usar funções dentro de funções, podemos ter funções "auxiliares", que nos ajudarão a alcançar um objetivo maior.

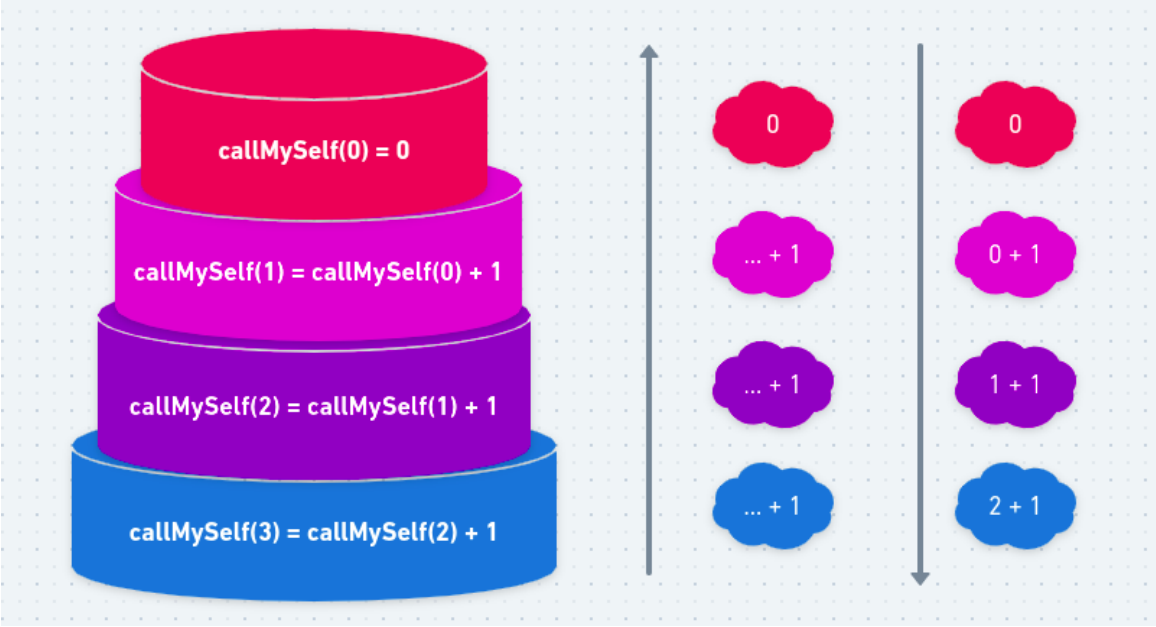
Por exemplo, podemos querer uma função que verifique se um número é primo e múltiplo de 3, nesse caso, poderíamos dividir as responsabilidades, e incumbir uma função de determinar se o número é primo, enquanto outra utilizaria desta função para verificar se o número é múltiplo de 3 e primo.

Maravilha

Porém, há casos em que uma função pode contar com ela mesma para alcançar seu objetivo, para isso, a função chama a si própria, criando assim o que chamamos de recursividade. Veremos como isso funciona na prática:

```
function callMySelf(times) {
  if (times === 0) {
    return 0
  }

  return callMySelf(times - 1) + 1
}

console.log(callmySelf(3)); // 3
```

O que aconteceu?




```
function callMySelf(times) {

  console.log("Start: " + times + " call");

  if (times === 0) {
    return 0
  }
```

```
  let nextCall = callMySelf(times - 1) + 1;

  console.log("End: " + times + " call");

  return nextCall;
}

console.log(callMySelf(3));

// Start: 3 call
// Start: 2 call
// Start: 1 call
// Start: 0 call
// End: 1 call
// End: 2 call
// End: 3 call
// 3
// [Finished in 0.1s]
```

O que acontece se não fizermos uma verificação?

```
function callMySelf(times) {

  console.log("Start: " + times + " call");

  let nextCall = callMySelf(times - 1) + 1;

  console.log("End: " + times);

  return nextCall;
}

console.log(callMySelf(3));

// RangeError: Maximum call stack size exceeded
```

Pensando recursivamente, basicamente para utilizarmos uma função dentro de outra, devemos sempre pensar em uma condição de parada, caso contrário a chamada continuará acontecendo até exceder o limite permitido

Para que serve recursividade?

Podemos utilizar recursividade para resolver problemas que dependem de instâncias menores do mesmo problema

E como utilizar recursividade? Bom basicamente precisamos analisar o problema e encontrar sua condição de parada, um caso em que haja uma solução bem definida, uma vez conhecida a condição de parada, podemos assumir que as outras condições do problema podem ser resolvidas e utilizar a própria função para isso;

Resumindo:

- Encontre a condição de parada e resolva: resolver o caso mais simples do problema
- Assuma que sua função funciona para casos menores
- Utilize os problemas menores para resolver seu problema original

Exemplos

- Encontrar o maior elemento de um array

  Você provavelmente conhece o método `Math.max()` em JavaScript, método que retorna o maior valor entre os números passados com parâmetro. Mas e se esse método não existisse, de que outra maneira poderíamos resolver este problema?

  Vamos pensar recursivamente!

  1. Condição de parada

     Qual o caso mais básico? Bom, supondo que a função sempre receberá um array, qual a situação em que seria mais fácil determinar o maior número? Bom, se nosso array só tiver um elemento, não precisaremos pensar muito, o maior elemento será justamente o único elemento, logo nossa condição de parada é: `array.length === 1`

     Ou seja

```
if (array.length === 1) {
  return array[0];
}
```

2. Instância menor

   Caso nosso array tenha mais de um elemento, qual seria o caso mais simples? Se só precisássemos decidir entre dois elementos, afinal um array de 2 elementos só é menos complexo que um array um único elemento. Sendo assim, uma instância menor do nosso problema original seria:

```
if (array.length === 2) {
  if (array[0] > array[1]) return array[0];
  else return array[1];
}
```

3. Problema original

   Para resolvermos o problema original, precisamos assumir que nossa função já existe e funciona para os problemas menores, logo como estamos querendo resolver parra um array [1, 2, 3, 4, 5] por exemplo, devemos assumir que a função já resolve o caso [2, 3, 4, 5].

   Dessa forma, se não nenhuma das condições anteriores for atendida, faremos o seguinte:

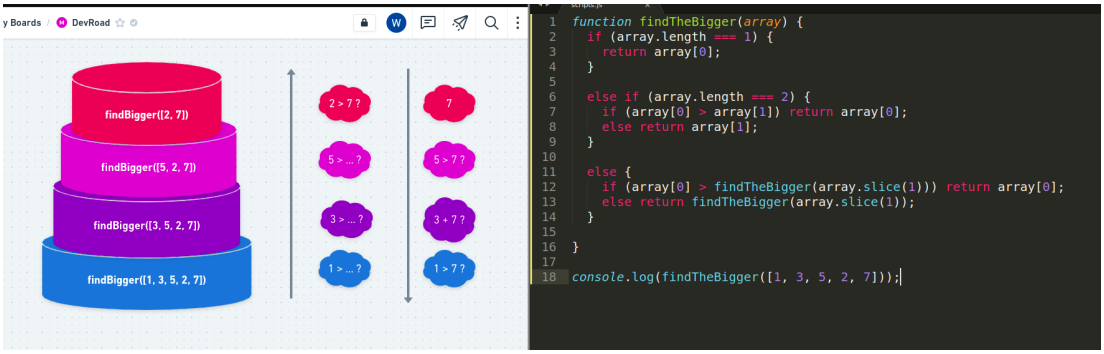
```
if (array[0] > callMySelf(array.slice(1))) return array[0];
else return callMySelf(array.slice(1));
```

Código completo ficará assim:

```
function findTheBigger(array) {
  if (array.length === 1) {
```

```
    return array[0];
  }

  else if (array.length === 2) {
    if (array[0] > array[1]) return array[0];
    else return array[1];
  }

  else {
    if (array[0] > findTheBigger(array.slice(1))) return array[0];
    else return findTheBigger(array.slice(1));
  }

}

console.log(findTheBigger([1, 20, 3, 7, 10, 15, 50]));
```

O fluxo se dará da seguinte forma:



- Reverter uma string

  No JavaScript temos uma função chamada .reverse() que simplesmente inverte a ordem dos caracteres de uma string, por exemplo "abc" vira "cba". Como poderíamos resolver isso usando recursão?

  Vamos aos passos:

  1. Condição de parada: Qual o momento em que devemos para nossa função? Haja vista que o problema estaria solucionado? Bem, caso a string seja vazia, "" ou tenha apenas uma letra "a" o retorno deverá ser a própria string. Nesse caso:

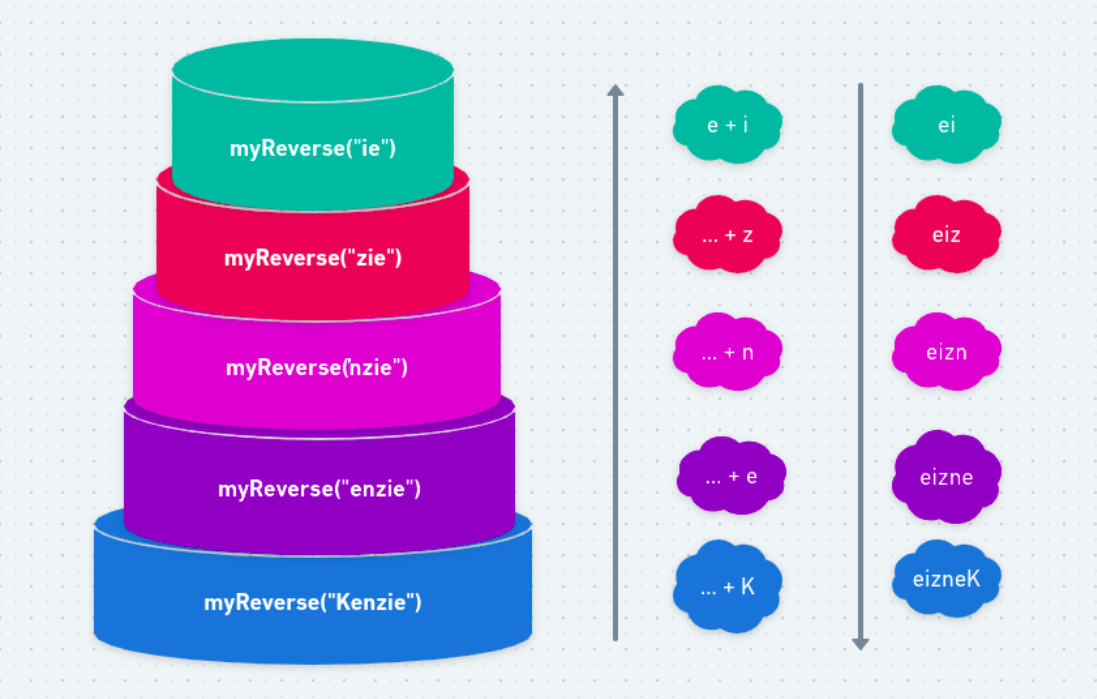
```
if (string.length <= 1) return string;
```

2. Agora, devemos buscar uma instância menor do nosso problema, por exemplo suponha a seguinte string `"Kenzie"`, uma instância menor desse problema seria a substring `"enzie"`, a menor em relação a essa seria `"nzie"` assim por diante.

Logo supondo que nossa função já existe e já resolve casos menores, podemos assumir que ao chamarmos `myReverse("enzie")`, o resultado será `"eizne"`. Para o caso da "Kenzie" tudo que precisaríamos fazer era inverter a ordem de retorno entre a primeira letra e o retorno da função `myReverse("enzie")`

Assim nosso código ficaria:

```
function myReverse(string) {
  if (string.length <= 1) return string;

  else  myReverse(string.slice(1)) + string[0];
}
```



- Seguir com mais exemplos:
  - Fibonacci

- Fatorial
- Primos