Programação Estruturada

Aula 13 - Recursão: Introdução, Números e Strings

Videoaula 02: Recursão e Números

Videoaula 02: Recursão e Números

Olá! Estou de volta para falar um pouco mais sobre a recursão. É verdade que identificar a natureza recursiva de um problema e encontrar uma solução exige um pouco de instinto do programador, porém, essa capacidade pode ser desenvolvida se você praticar bastante. Nesta videoaula, você verá alguns exemplos de problemas que podem ser resolvidos através da recursão, e continuarei explicando vários tópicos e ideias que te ajudarão a praticar alguns deles. Por enquanto, vamos nos ater a problemas que envolvam apenas números, ok? Vamos lá!

Provavelmente, o exemplo mais famoso de recursão é o cálculo do fatorial de um número natural. A definição dessa função pode ser vista no slide (Figura 1). Se `n` for igual a `0`, o `fatorial(n)` é `1`. Caso contrário, se `n` for maior que `0`, o `fatorial(n)` é igual a `n` vezes o fatorial de `n-1`.

Figura 1 - Fatorial

Seja n número natural, então:

```
fatorial(n) = 1                  , se n = 0
fatorial(n) = n * fatorial(n-1)  , se n > 0
```

Nessa fórmula de cálculo para o fatorial de `n`, você notou algo diferente?

Observe que o fatorial de `n` é definido em termos do fatorial de `n - 1`, ou seja, temos uma definição recursiva! Baseado nessa nova definição, veja o cálculo mostrado a seguir (Figura 2).

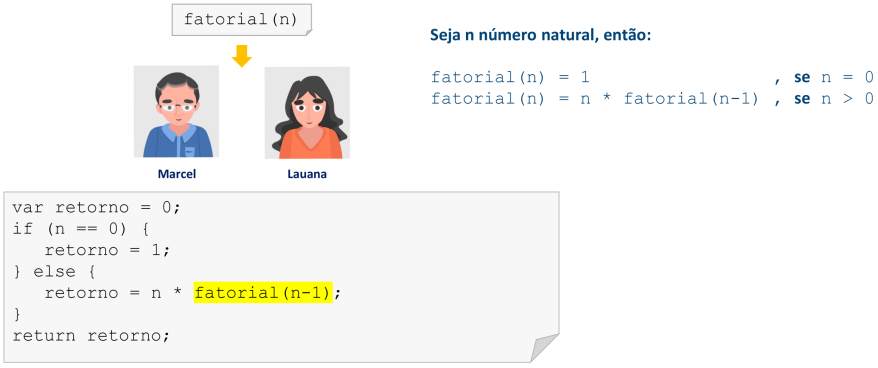
Figura 2 - Fatorial

```
fatorial(3)
= 3 * fatorial(3-1)
= 3 * fatorial(2)
= 3 * (2 * fatorial(2-1))
= 3 * (2 * fatorial(1))
= 3 * (2 * (1 * fatorial(0)))
= 3 * (2 * (1 * 1))
= 3 * (2 * 1)
= 3 * 2
= 6
```

Note que para calcular o fatorial de 3, é necessário calcular o fatorial de (3 – 1), que corresponde ao fatorial de 2, cujo resultado depende do valor do fatorial de 1, e assim por diante, até chegarmos ao fatorial de 0, já definido como 1, que é o **caso base** dessa função. Nesse momento, a função começa a dar os devidos retornos até chegarmos ao produto final, resultando no valor 6 para o fatorial de 3.

Agora, que tal vermos como aconteceria o cálculo recursivo de `fatorial(3)` usando os bilhetes, assim como fizemos anteriormente? A mensagem escrita no bilhete será uma tradução exata da definição de uma função fatorial: Se n for 0, o retorno é 1. Caso contrário, o retorno é `n*fatorial(n-1)`. Neste caso, se n > 0, teremos a recursão e a chamada será enviada para outra pessoa.

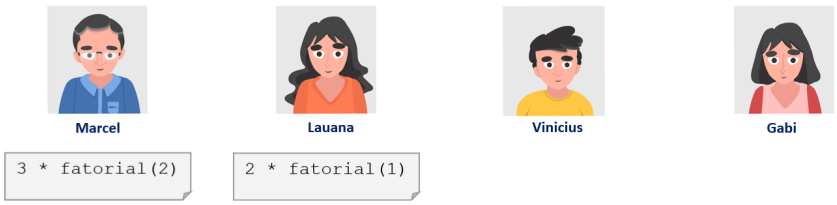
Figura 3 - Passando Bilhetes...



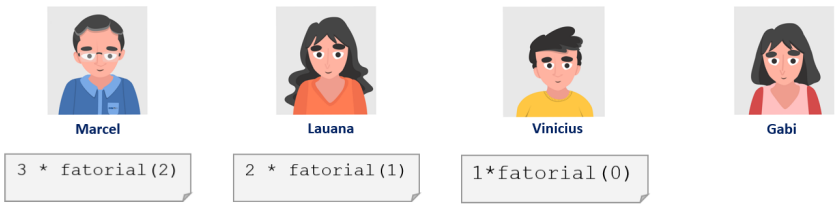
Precisamos novamente de Marcel, Lauana, Vinicius e Gabi.

Inicialmente, Marcel recebe o bilhete `fatorial(3)`

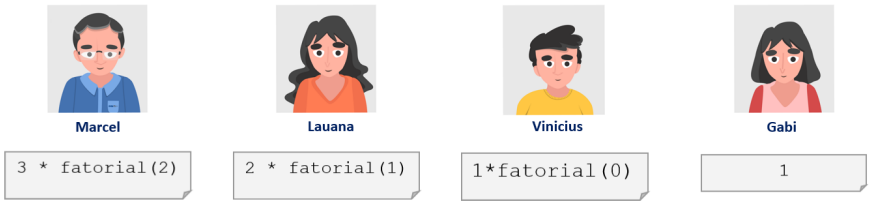
E começa a escrever o seu bilhete. No entanto, ele nota que precisa gerar o `fatorial(2)`. Como ele não pode fazer isso, pois já está gerando `fatorial(3)`, escreve um bilhete `fatorial(2)`



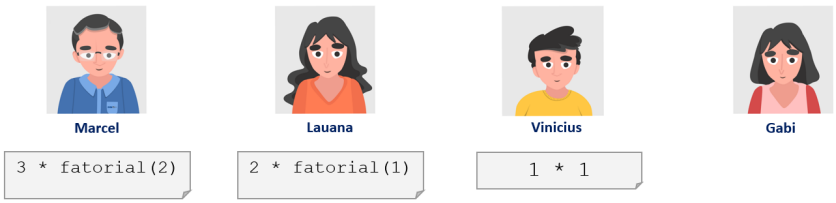
E passa para Lauana, que recebe esse bilhete. E começa a escrever o seu bilhete. Lauana, então, nota que precisa gerar o `fatorial(1)`. Como não pode fazer isso, ela escreve um bilhete `fatorial(1)`



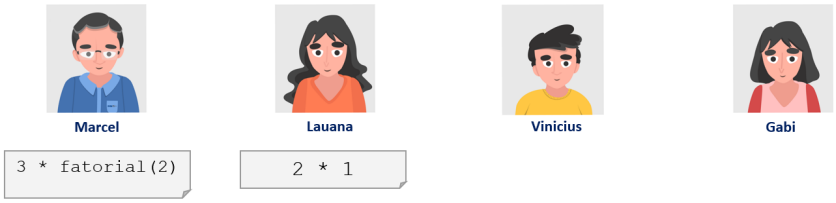
E passa para Vinícius, que recebe esse bilhete. E começa a escrever o próprio bilhete. Vinícius percebe que precisa gerar o `fatorial(0)`. Como não consegue, ele escreve um bilhete `fatorial(0)`



E passa para Gabi, que recebe esse bilhete. E começa a escrever o próprio bilhete. O bilhete de Gabi, porém, pode ser escrito sem a ajuda de ninguém, pois o `n` passado foi 0. Por isso, Gabi consegue terminar o seu bilhete contendo o valor 1.

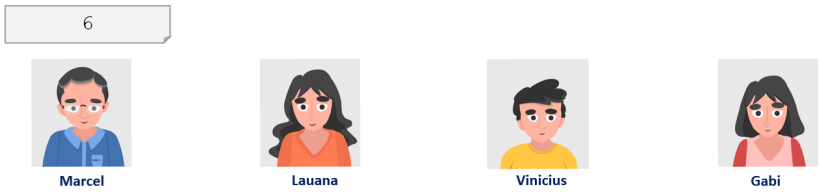


Gabi então passa esse bilhete para Vinícius, que substitui no seu bilhete a chamada do `fatorial(0)` pelo conteúdo do bilhete dela.



Vinicius então passa esse bilhete para Lauana, que substitui no seu bilhete da chamada do `fatorial(1)` pelo conteúdo do bilhete dele. Para simplificar, ele já repassou o resultado da expressão `1 * 1`. O mesmo acontecerá com os demais.

Lauana, então, passa esse bilhete para Marcel, que substitui no seu bilhete a chamada do `fatorial(2)` pelo conteúdo do bilhete dela.



E, finalmente, Marcel retorna o conteúdo do seu bilhete, o qual contém a resposta à chamada `fatorial(3)`.

Legal, né? Vamos ver como fica essa função no código JavaScript?

Código 1 - 13_3 Fatorial Recursivo.html e 13_3 Fatorial Recursivo.js

```html
<html >
   <head>
      <meta charset="UTF-8" />
      <title>Programação Estruturada - Aula 13</title>
   </head>
   <body>
      <noscript>Seu navegador não suporta JavaScript ou ele está desabilitado.</noscript>

      <h1>Fatorial Recursivo</h1>

      N: <input type="number" id="numero" value="">
      <button onclick="exibir()">Fatorial</button>
      <br>
      <p id="resultado"></p>

      <script src="script.js"></script>
   </body>
</html>

```

```javascript
function exibir() {
   var n = Number(document.getElementById("numero").value);
   var resultado = n + "! = " + fatorial(n);
   document.getElementById("resultado").innerHTML = resultado;
}

function fatorial(n) {
   var retorno = "";

```

```
10    // Condição de TERMINAÇÃO
11    if (n < 0) {
12       retorno = "ERRO";
13
14    // CASO BASE
15    } else if (n == 0) {
16       retorno = 1;
17
18    // RECURSÃO (n > 0)
19    } else {
20       retorno = n * fatorial(n-1);
21    }
22    return retorno;
23 }
24
```

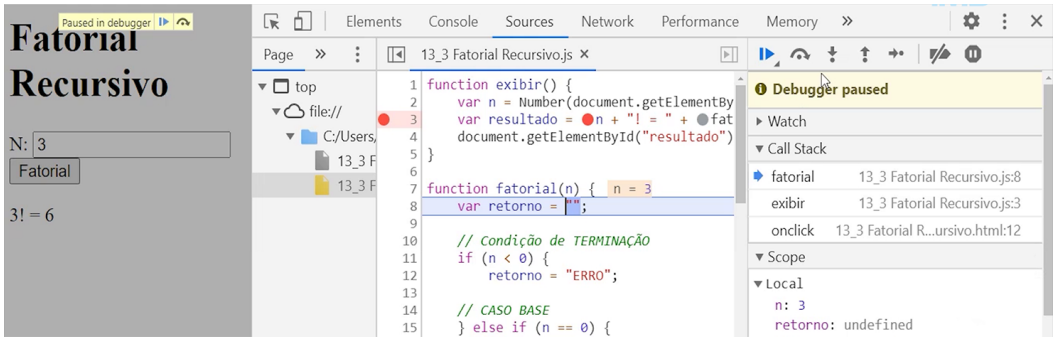
Na página HTML, ao inserirmos 3, por exemplo, teremos o fatorial de 3 e será impresso 3! = 6. Isso foi feito com uma função recursiva, ok? Vamos conhecer essa função. Veja que a página é muito simples, parecida com a do exemplo anterior, ao qual algum número deverá ser inserido. A diferença é que agora será exibido o fatorial. No JavaScript, a função `exibir` é praticamente a mesma que vimos no exemplo anterior, só que agora a variável `resultado` é `n!`, e chamaremos a função `fatorial` passando o `n` que foi lido.

Agora, iremos para a função recursiva do cálculo do fatorial. Essa função tem: uma **condição de terminação**, não existe fatorial de número negativo, então se o número for negativo, ou seja, se o `n` for menor que 0, será retornado o texto `"ERRO"`; tem o **caso base**: se o `n` for igual a 0, será retornado 1; e tem o **caso recursivo**, que é a condição `else`, ou seja, o `n` não é menor que 0 e não é igual a 0, ele só pode ser maior que 0. O retorno será exatamente a definição da função, `n * fatorial(n-1)`. Por fim, retornamos o texto do resultado.

Você pode ir à página HTML para realizar alguns testes, por exemplo, o que acontece se for pedido o fatorial de -1, de fato, será retornado "ERRO"; o fatorial de 0 retornará o 1; e o fatorial de 3, retornará o 6.

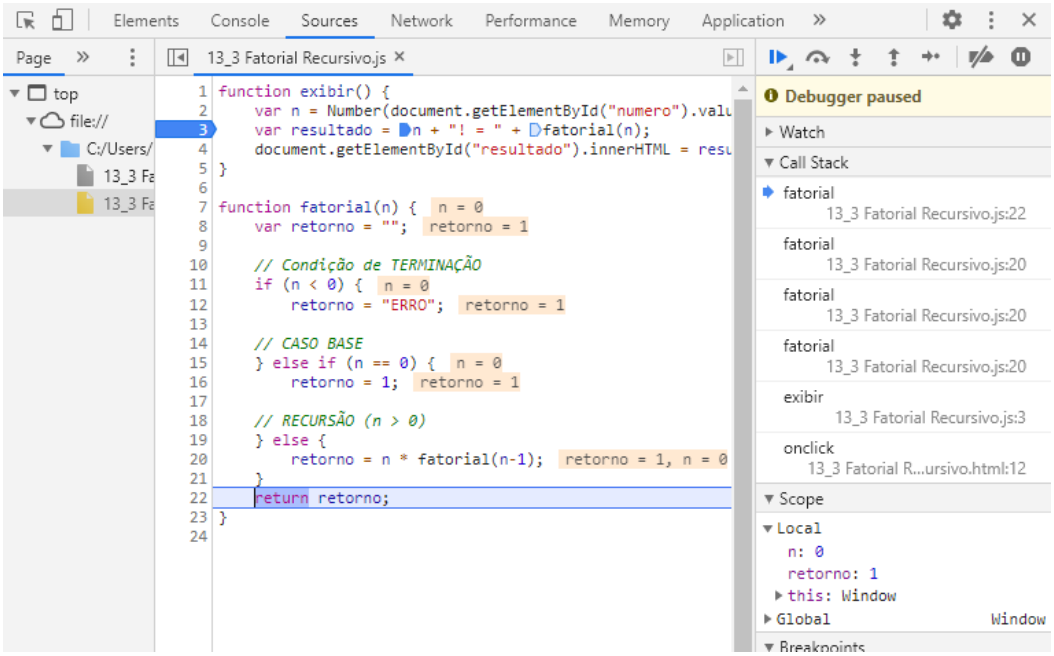
Abrimos a ferramenta de depuração da página, colocamos um *breakpoint* na linha 3, fazemos um *step into*, entramos na função `fatorial`, com `n` igual a 3, e no *Call Stack*, temos uma chamada a `fatorial`. Veja a Figura 4.

Figura 4 - Depuração da Execução do Fatorial



Entramos no **caso recursivo**, voltamos com `n` igual a 2 e empilhamos mais uma chamada a `fatorial`. Entramos novamente para o **caso recursivo**, porque `n` é igual a 2, que vai chamar `fatorial` novamente, e então empilhamos mais uma chamada a `fatorial` com `n` igual a 1. E, mais uma vez, iremos para o **caso recursivo**, porque `n` é maior que 0, e faremos mais uma chamada com `n` igual a 0, e teremos no final as quatro chamadas à `fatorial`. Temos agora um **caso base**, em que o `n`, de fato, é igual a 0 e o `retorno` é igual a 1, como exposto na Figura 5. E ele criará *return value* (valor de retorno) igual a 1, no escopo.

Figura 5 - Caso Base na Depuração



Iremos retornar e estaremos com escopo onde `n` é igual a 1. O `retorno` será 1, porque é o resultado de 1 * 1. Criamos o valor de `retorno`, desempilhamos mais uma chamada e voltamos para o contexto onde o `n` é igual a 2. Faremos 2 * 1, que é o que veio da última chamada. Construímos o valor de retorno igual a 2, desempilhamos mais uma chamada e o retorno será 6 por conta da expressão 3 * 2, em que o `n` é igual a 3. Desempilhamos a chamada e voltamos para função `exibir` e o resultado 6 vai ser exibido na página HTML. Ok?

Na videoaula anterior, mencionei o fato de que recursão e iteração têm o mesmo poder de expressão e podem ser substituídas uma pela outra. No caso do fatorial, veja que podemos começar com o valor do **caso base**, ou seja, 1, e fazer um laço em `i`, variando entre `1` e `n`. Observe que, se `n` for `0`, o laço para imediatamente com a resposta já definida.

Vejamos como fica por meio desse exemplo (Código 2).

Código 2 - 13_4 Fatorial For.html e 13_4 Fatorial For.js

```html
<html >
   <head>
      <meta charset="UTF-8" />
      <title>Programação Estruturada - Aula 13</title>
   </head>
   <body>
      <noscript>Seu navegador não suporta JavaScript ou ele está desabilitado.</noscript>

      <h1>Fatorial Iterativo</h1>

      N: <input type="number" id="numero" value="">
      <button onclick="exibir()">Fatorial</button>
      <br>
      <p id="resultado"></p>

      <script src="script.js"></script>
   </body>
</html>

```

```javascript
function exibir() {
   var n = Number(document.getElementById("numero").value);
   var resultado = n + "! = " + fatorial(n);
   document.getElementById("resultado").innerHTML = resultado;
}

function fatorial(n) {
   var retorno = "";

   if (n < 0) {
      retorno = "ERRO";
   } else {
      var fatorial = 1;
      for(var i = 1; i <= n; i++) {
         fatorial = fatorial * i;
      }
      retorno = fatorial;
   }
   return retorno;
}

```

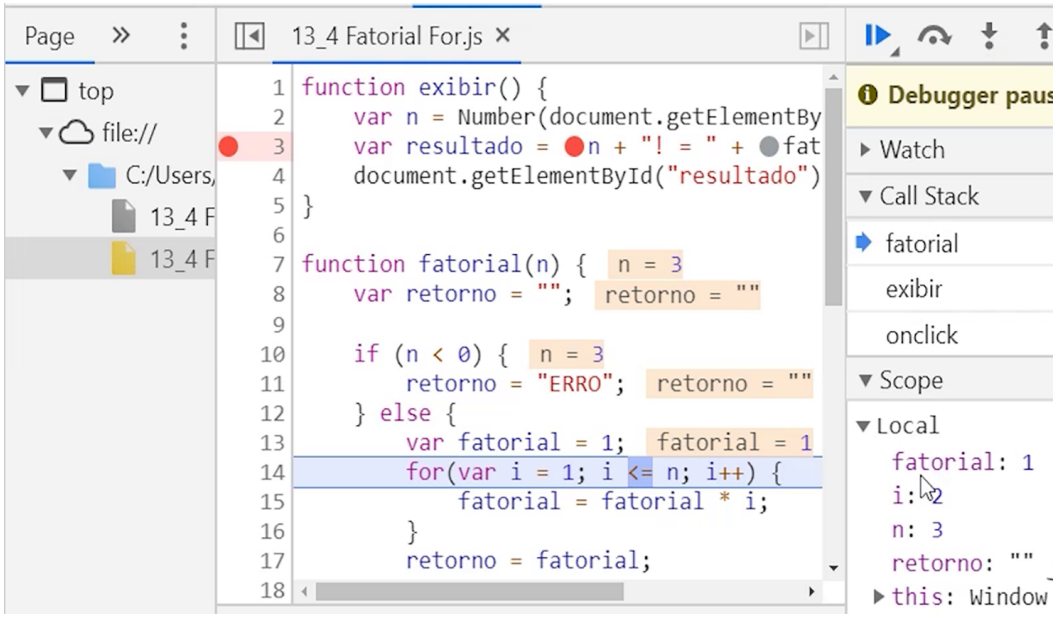
No exemplo dado, o HTML é o mesmo, não tem nenhuma mudança. A mudança foi no JavaScript, onde também a função `exibir` é exatamente a mesma. A diferença fica apenas na função `fatorial`, que não é mais recursiva, logo você poderá olhar todo o

corpo da função `fatorial` e não encontrará a chamada à função `fatorial`.

Veja que foi utilizada uma variável chamada `fatorial`, mas a função `fatorial` não está sendo chamada. Temos `retorno`, e aquele caso de erro que tínhamos na função recursiva é mantido: se `n` for menor que 0 aparecerá a mensagem `"ERRO"`. Caso contrário, ao invés de fazermos uma solução recursiva, faremos uma solução iterativa, começando com `fatorial` igual a 1, em que qualquer número multiplicado por 1 será ele mesmo. Observe que foi feito um laço com `i` variando de 1 até o `n` (linha 14) e faremos `fatorial` ser igual ao `fatorial * i` (linha 15). Então se tivermos o fatorial de 3, por exemplo, teremos 3 * 2 * 1 * 1, que é o **caso base**, e o `retorno` será exatamente esse fatorial, 6.

Então se formos à página HTML e fizermos o fatorial de 3, o resultado será o mesmo e poderemos até fazer novamente uma depuração, colocando o *breakpoint* na linha 3. Veremos que a *call stack*, à medida que formos avançando na função `fatorial`, não aumentará. Temos `n` é igual a 3, então entraremos na condição `else` em que a variável `fatorial` é igual a 1, entraremos no laço `for`, onde tem o `i` começando com 1, e enquanto esse `i` for menor ou igual a `n`, esse `fatorial` será incrementado, então o `i` aumentará para 2 e o `fatorial` será 1 * 1, que é 1. Depois de passar dessa linha 15 (Figura 7), `fatorial` será mudada para 2 e o `i` para 3, e depois mudamos `fatorial` para 6. Note que em momento algum aumentaremos essa chamada, então não usaremos a recursão, e, portanto, a pilha não será incrementada. Essa é uma solução iterativa do fatorial.

Figura 7 - Depuração do `fatorial` Iterativo



Outro exemplo clássico de programa recursivo é aquele que exibe a sequência de Fibonacci. Você lembra que sequência é essa? Ela começa com os números 0 e 1. A partir daí, o próximo número será sempre a soma dos dois números anteriores. Veja no slide (Figura 8) os 10 primeiros números da sequência Fibonacci. Note que a posição 0 e a posição 1 são ocupadas pelos elementos 0 e 1, respectivamente. Já a posição 2, é o resultado da soma dos dois anteriores, ou seja, 0 + 1. Continuando a sequência, a posição 3 tem valor 2. Esse é o resultado da soma dos elementos que estão na posição 1 e 2, ou seja, o resultado de 1 + 1. Seguimos assim até chegarmos à posição 9, que é ocupada pelo elemento 34, e o resultado é a soma de 13 com 21, que são, respectivamente, os valores das posições 7 e 8.

Figura 8 - Sequência de Fibonacci

```
fibonacci(n) = 0                                , se n = 0
fibonacci(n) = 1                                , se n = 1
fibonacci(n) = fibonacci(n-2) + fibonacci(n-1) , se n > 1
```

0	1	2	3	4	5	6	7	8	9
0	1	1	2	3	5	8	13	21	34

A sequência de Fibonacci aparece com frequência na matemática. Seus números são tão importantes que existe uma revista inteira dedicada ao estudo deles, a *Fibonacci Quarterly*.

Curiosidade

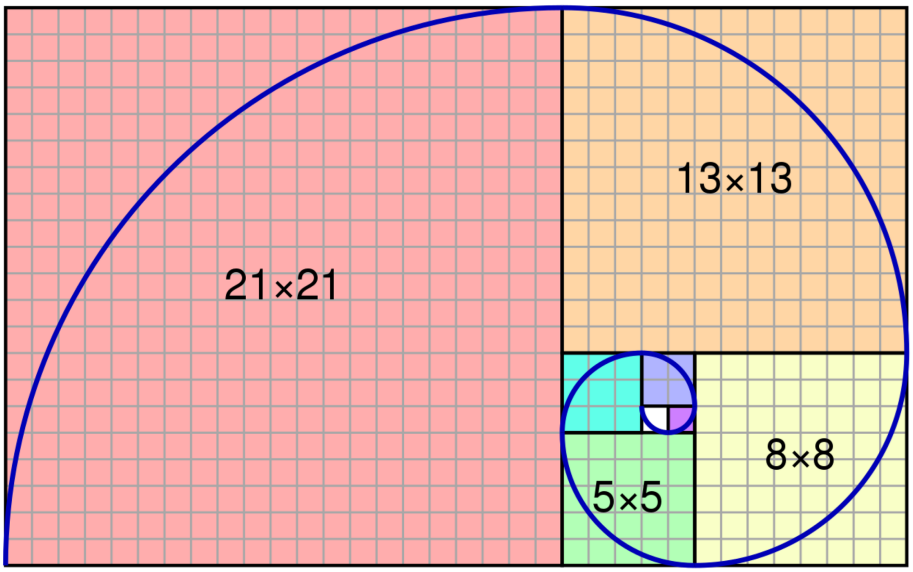
Veja no link abaixo:

URL: The Fibonacci Quarterly

As aplicações dessa sequência incluem algoritmos, como a técnica de pesquisa Fibonacci, e estruturas de dados como o *heap* de Fibonacci e grafos chamados de cubos de Fibonacci, que são usados para interconectar sistemas paralelos e distribuídos.

Observe a imagem (Figura 9). Este é o espiral de Fibonacci: uma aproximação da espiral dourada criada pelo desenho de arcos circulares que conectam os cantos opostos dos quadrados no mosaico de Fibonacci, ou seja, "um mosaico com quadrados cujos comprimentos laterais são números sucessivos de Fibonacci: 1, 1, 2, 3, 5, 8, 13 e 21".

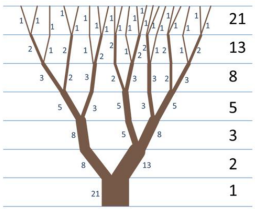
Figura 9 - Espiral de Fibonacci



FONTE: https://en.wikipedia.org/wiki/Fibonacci_number – 01/04/2020

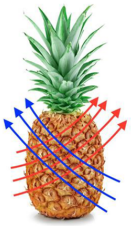
A sequência de Fibonacci também aparece na Biologia. Por exemplo, dentre várias aparições, podemos destacar que ela está presente na ramificação de árvores, no arranjo de folhas em um caule e nos brotos de um abacaxi. E aí? Interessante, não acha?

Figura 10 - Sequência de Fibonacci na natureza



FONTE: https://tugofweb.com/2015/04/07/fibonacci-trees/ – 20/08/2020

FONTE: https://www.eanda.com.br/blog/didatica-metodologia-golden-ratio-e-voce/ – 20/08/2020



FONTE: https://twitter.com/mrsluchies/status/914576686850723840– 20/08/2020

Veja um exemplo (Código 3) da implementação da função `fibonacci` usando recursão. Note que teremos dois **casos base**, 0 e 1. Lembre-se que a posição `fibonacci(9) = 34`.

Código 3 - 13_5 Fibonacci.html e 13_5 Fibonacci.js

```html
<html >
   <head>
      <meta charset="UTF-8" />
      <title>Programação Estruturada - Aula 13</title>
   </head>
   <body>
      <noscript>Seu navegador não suporta JavaScript ou ele está desabilitado.</noscript>

      <h1>Fibonacci</h1>

      N: <input type="number" id="numero" value="">
      <button onclick="exibir()">Fibonacci</button>
      <br>
      <p id="resultado"></p>

      <script src="13_5 Fibonacci.js"></script>
   </body>
</html>

```

```javascript
function exibir() {
   var n = Number(document.getElementById("numero").value);
   var resultado = "fibonacci(" + n + ") = " + fibonacci(n);
   document.getElementById("resultado").innerHTML = resultado;
}

function fibonacci(n) {
   var retorno = "";

   // Condição de TERMINAÇÃO
   if (n < 0) {
      retorno = "ERRO";

   // CASO BASE
   } else if (n == 0) {
      retorno = 0;

   // CASO BASE
   } else if (n == 1) {
      retorno = 1;

   // RECURSÃO (n > 0)
   } else {
      retorno = fibonacci(n-2) + fibonacci(n-1);
   }

```

```
27    return retorno;
28 }
29
```

Nesse exemplo, a página HTML é praticamente idêntica ao HTML do fatorial, que recebe apenas um número, e queremos imprimir o elemento da sequência Fibonacci começando pelo 0, que é o primeiro elemento, o 1, que é o segundo elemento, e assim por diante. Como o HTML é praticamente o mesmo, não tem mais o que apresentar. O nosso arquivo JavaScript, ele tem a definição da função `Fibonacci`. Note que a função `exibir`, novamente, é praticamente idêntica ao exemplo do fatorial, e basicamente tudo que fazemos é pegar o valor do número na linha 2 e na linha 4 escrever o resultado num parágrafo que temos no HTML. Veja também que na linha 3 temos a geração do texto que será exibido: `"fibonacci (" + n + " ) = " + fibonacci(n)`.

Focaremos na função `Fibonacci`. Essa função recebe um `n` e ela tem uma variável `retorno`, que é o que será retornado por ela. Ela tem uma **condição de terminação**: se o `n` for menor que 0, não existe Fibonacci de um número negativo, então se o `n` for menor que 0, será retornado o texto `"ERRO"`. Caso contrário, entramos na condição `else`. Se `n` for igual a 0, temos um **caso base**, e se o `n` for igual a 1, temos outro **caso base**. Assim, essa função tem dois casos bases: quando o `n` é igual a 0 e quando o `n` é igual a 1.

Nas linhas 15 e 16, temos o primeiro **caso base**, que é se o `n` for 0, o retorno é 0. Nas linhas 19 e 20, temos o segundo **caso base**, senão, que é se o `n` for 1, o retorno é 1. Por fim, se o `n` não é menor que 0, se não é igual a 0 e se não é igual a 1, ele só pode ser maior do que 1 que é o **caso recursivo** que está entre as linhas 23 e 24. E nesse caso, o que fazemos é retornar a soma dos últimos dois elementos. Então se o `n` for 2, por exemplo, teremos a soma de Fibonacci de 2 - 2, que é 0, e Fibonacci de 2 - 1, que é 1. Dessa forma, estaremos somando o Fibonacci de 0 e o Fibonacci de 1 que, por acaso, são **dois casos base**. O Fibonacci de 0 sendo 0, e o Fibonacci de 1 sendo 1, ou seja, 0 + 1 é 1. Você poderá ver na página HTML que se tivermos o Fibonacci de 3, por exemplo,

o Fibonacci de 3 será 2, que é 1 + 1, o Fibonacci de 4 é 3, que é 2 + 1, até chegarmos ao lembrete dado antes de iniciar esse exemplo, de que Fibonacci (9) é igual 34. É assim que fazemos de maneira recursiva a construção da sequência de Fibonacci.

Concluímos assim mais uma videoaula. No nosso próximo encontro, veremos como utilizar recursão em *strings*. Até breve!