Aula Virtual 1 de Abril

April 1, 2020

0.1 Aula de primeiro de Abril

A ideia da aula de hoje é de se mostrar que há outra forma de se fazer um número arbitrário de repetições sem usar recursão, mas com o uso de laços.

O primeiro a ser visto é o **while** ou seja enquanto, na sua forma básica ele é da seguinte forma
while condição
comandos
end
Ou seja, enquando a condição for válida, os comandos são executados. Vamos a um exemplo:

```
In [ ]: function regressiva(n)
            while n > 0
                print(n, " ")
                print(" . ")
                n = n - 1
            end
            println("Fim")
        end
        regressiva(10)
```

Claro que a condição de parada pode ser mais complexa e o número de comandos dentro do corpo do **while** pode ser variado, e sim, nada impede de se ter um **while** dentro de outro **while**

```
In [ ]: function regressivalenta(n)
            while n > 0
                print(n, " ")
                i = 10
                while i > 0
                    print("loading ")
                    sleep(10)
                    i = i - 1
                end
                println("...")
                n = n - 1
            end
            println("Fim")
        end

In [ ]: regressivalenta(8)
```

Para continuar, vamos refazer os mesmos exercícios da aula anterior, agora usando o **while**. São eles: * Verificar se um número é primo * Imprimir os n primeiros primos * Imprimir os n primeiros primos de Mersenne

```
In [4]: function testedivisores()
            if ndivisores(14) != 4
                println("Não funcionou para 14: ",ndivisores(14))
            end
            if ndivisores(16) != 5
                println("Não funcionou para 16: ", ndivisores(16))
            end
            if ndivisores(1) != 1
                println("Não funcionou para 1: ", ndivisores(1))
            end
            if ndivisores(1024) != 11
                println("Não funcionou para 1024", ndivisores(1024))
            end
            if ndivisores(0) != 0
                println("Não funcionou para zero", ndivisores(0))
            end
            if ndivisores(13) != 2
                println("Não funcionou para 13", ndivisores(13))
            end
            println("Final dos testes")
        end


        function ndivisores(n)
            divi = 0
            i = 1
            while i < sqrt(n)
                if n % i == 0
                    divi += 1
                end
                i += 1
            end
            divi *= 2
            if i == sqrt(n)
                divi += 1
            end
            return divi
        end

        function eprimo(n)
            if ndivisores(n) == 2
                return true
            else
                return false
```

```
            end
        end

        function imprimenprimos(x)
            i = 0
            candidato = 1
            while i < x
                if eprimo(candidato)
                    println(candidato)
                    i += 1
                end
                candidato += 1
            end
        end

        function imprimenmersenne(x)
            i = 0
            pot = 0
            while i < x
                if eprimo(2^pot - 1)
                    println("2^", pot, "-1 é primo: ", 2^pot -1)
                    i += 1
                end
                pot += 1
            end
        end
```

```
Out[4]: imprimenmersenne (generic function with 1 method)
```

Escolha um dos exercícios de repetição das aulas anteriores para fazermos com o **while**

```
In [5]: imprimenmersenne(8)
```

```
2^2-1 é primo: 3
2^3-1 é primo: 7
2^5-1 é primo: 31
2^7-1 é primo: 127
2^13-1 é primo: 8191
2^17-1 é primo: 131071
2^19-1 é primo: 524287
2^31-1 é primo: 2147483647
```

```
In [16]: function test()
             if fibo2(1) != 1 ||
                    fibo2(2) != 1||
                    fibo2(5) != 5||
                    fibo2(10) != 55 ||
                    fibo2(50) != 12586269025
```

```
            return "Erro. Verifique sua função fibo2(n)"
        end
        return "Final dos testes"
    end

    function fibo2(n)
      a = 1
      b = 1
      i = 1
      while i < n
        temp = a + b
        a = b
        b = temp
        i += 1
      end
      return a
    end

    test()
    fibo2(50)
```

```
Out[16]: 12586269025
```

Como vocês devem ter percebido um while que começa em 1 e vai até n, de um em um é muito usual. Logo há um comando que resume isso, o **for**

```
In [18]: for j = 3:8
             println(j)
         end
```

```
3
4
5
6
7
8
```

Historicamente, o **for** foi criado para acompanhar operações com vetores e matrizes, mas é usado em diversas situações atualmente. Em Julia o comando **for** voltou as origens

Agora escolha um exercício para resolvermos com o **for** :)

```
In [20]: function test()
             if fibo2(1) != 1 ||
                     fibo2(2) != 1||
                     fibo2(5) != 5||
                     fibo2(10) != 55 ||
                     fibo2(50) != 12586269025
                 return "Erro. Verifique sua função fibo2(n)"
```

```
        end
        return "Final dos testes"
    end

    function fibo2(n)
      a = 1
      b = 1
      i = 1
      for i in 1:n-1
        temp = a + b
        a = b
        b = temp
      end
      return a
    end
    test()
```

```
Out[20]: "Final dos testes"
```

Há dois comandos usados em laços, o **break** e o **continue** vamos ver como eles funcionam no **for** abaixo:

```
In [22]: for i in 1:20
             if i > 10
                 break
             end
             if i % 2 == 0
                 continue
             end
             println(i)

         end
```

```
1
3
5
7
9
```

Mas, notem que o uso de **continue** e **break** nem sempre ajuda a entender o código.