Alfredo's MAC0110 Journal

Alfredo Goldman

April 12, 2020

0.0.1 Agora na linha de comando

Hoje, vamos continuar fazendo exercícios com Julia, a diferença é que agora vamos usar a linha de comando, isso é, vamos criar arquivos fora do ambiente "seguro" do Jupyter. Para isso, vamos criar arquivos com a extensão .jl com código. Vamos começar criando um arquivo imprime.jl com apenas um print

```
println("Agora do arquivo")
```

Ele pode ser compilado/executado chamando se julia imprime.jl

Da mesma forma podemos usar arquivos para guardar funções como a de cálculo de potências inteiras, que recebe um valor x, e devolve x^n.

```
function pot(x, n)
  res = 1
  while n > 0
    res = res * x
    n = n - 1
  end
  return res
end
```

Mas, como podemos usar essa função, agora que ela está pronta? Usando o comando include, há formas mais sofisticadas de usar "pacotes", mas por enquanto esse comando será suficiente.

```
include("funct.jl")
println("2 elevado a 4 eh ", pot(2, 4)
```

Isso inclusive nos ajuda no que se refere aos testes automatizados, pois o arquivo com os testes automatizados pode ser executado de forma independente.

Sim, podemos incluir novas funções no arquivo funct.jl, como o cálculo de fatorial.

Os testes de potência a fatorial podem ser independentes!

Agora que temos as funções de cálculo de potência e de fatorial, podemos usá-las para cálculos mais sofisticados como o de cosseno, usando séries de Taylor:

$$\cos(x) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!}$$

, sendo que o valor de x é dado em radianos.

```
include("funct.jl")
function cosseno(x)
  Erro = 1f-7
  serie = 0
  termo = 1
  i = 0
  while abs(termo > Erro)
     serie = serie + termo
     i = i + 1
     termo = potencia(-1, i) * potencia(x, 2 * i) / fat(2 * i)
  end
  serie = serie + termo
  return serie
end
```

Mas, podemos melhorar a nossa função de cálculo de cosseno observando que dá para a partir do termo anterior, chegar ao próximo termo.

```
# aqui vai o codigo
```

Agora uma discussão sobre uma dúvida que apareceu para o monitor, a identação. Isso é o recuo que fazemos para delimitar blocos de código em Julia. Vamos ver um exemplo:

```
function matematica()
   i = 1
   while i < 100
      if i % 2 == 0
```

```
            println(i, " eh par ")
        else
            if i % 3 == 0
                print(i, " eh divisivel por 3 e ")
                soma = 0
                aux = i
                while aux > 0
                    soma = soma + aux % 10
                    aux = div(aux, 10)
                end
                if soma % 3 == 0
                    println(" e a soma dos seus digitos tambem")
                else
                    println("Deu ruim, resultado nao esperado para: ", i)
                    break # sim esse break e justificado :)
                end
            end
        end
        i = i + 1
    end
end
```

Além dos blocos, também é interessante entender o conceito de escopo, vamos a um exemplo:

```
function valeum(a)
    println("a valia: ", a)
    a = 1
    println("agora a vale ", a)
end
function avaleum()
    # nao imprime a aqui, pois daria erro
    a = 1
    println("a vale: ", a)
end

function vamosver()
    a = 3
    println("a vale: ", a)
    begin
      println("Modifiquei a em um bloco")
      a = 2
      println("a vale: ", a)
    end
    println("O a vale, fora do bloco ", a)
    valeum(a)
    println("a vale: ", a)
    avaleum()
    println("a vale: ", a)
end
```

No código acima, podemos ver duas coisas importantes, o escopo (valor de uma variável vai além dos blocos, pois ao modificar dentro de um bloco, modificamos a variável original. Por outro lado, o escopo é independente conforme a função.

Ao chamar uma função com uma variável é passada uma cópia da variável, que é no início da função igual ao valor original.

Um exercício para terminar. Dado um número n sabe-se que n^3 pode ser representado pela soma de n números ímpares consecutivos, encontre esses valores para n de 1 a 10.