Alfredo's MAC0110 Journal

Alfredo Goldman

April 21, 2020

1 Programa do curso

1.1 Aula 11 - *<2020-04-22 qua>*

1.1.1 Entrada de dados e o começo de listas

Nessa aula, temos dois tópicos principais, como fazer a entrada de dados, através de comandos de entrada e com argumentos. Além disso também veremos como tratar de um tipo especial de variável, onde é possível, guardar mais de um valor.

1. O comando input Quando queremos inserir dados, em Julia, tanto no Jupyter, como no mode interativo, basta colocar dados. Mas, como podemos fazer para entrar dados em um programa comum?

   Para isso temos o comando readline(), que interrompe a execução do programa e espera pela entrada de uma String, o que ocorre quando a tecla <enter> é pressionada.

   ```
   println("Digite o seu nome")
   resposta = readline()
   println("O seu nome e: ", resposta)
   ```

   Como o readline() lê Strings, se quisermos ler números, é necessário usar o comando parse.

   ```
   println("Digite um inteiro")
   valor = parse(Int64, readline())
   println("O numero digitado foi ", valor)
   ```

   Sabendo ler números do teclado, vamos a um exercício simples, ler uma sequência de números inteiros terminada por zero e devolver a sua soma.

   ```
   # coloque a sua proposta aqui
   ```

2. Lendo através da linha de comando A outra forma de ler comandos é através da constante ARGS que é preparada na chamada de um programa. Para entender melhor isso, vamos ver o seguinte programa.

   ```
   println(ARGS)
   ```

   Se a linha acima está no arquivo args.jl, ao chamar julia args.jl com diversos parâmetros, teremos diversos resultados diferentes.

   Por exemplo ao chamar:

   julia args.jl 1 2 3 abc

   Teremos como resposta

   ```
   ["1", "2", "3", "abc"]
   ```

   Vamos analisar um pouco melhor essa resposta observando que cada parâmetro está em uma posição.

   ```
   tam = length(ARGS)
   println("O tamanho dos argumentos e: ", tam)
   for i in 1:tam
     println(ARGS[i])
   end
   ```

   Olhando o código acima, podemos ver que o comando length() devolve o número de argumentos, ou seja, o tamanho da lista ARGS. Além disso com os colchetes é possível acessar a cada posição da lista de forma individual.

   O exemplo abaixo soma os parâmetros inteiros dados como argumentos. Ele também ilustra uma boa prática que é, sempre colocar o código em módulos, no caso abaixo em funções:

```
function main()
  tam = length(ARGS)
  s = 0
  i = 1
  while i <= tam
      valor = parse(Int, ARGS[i])
      println(valor)
      s = s + valor
      i = i + 1
  end
  println("A soma foi: ", s)
end
main()
```

A flexibilidade que temos ao usar listas é enorme! Por isso, listas ou vetores, merecem um tópico próprio.

3. Listas

Vamos primeiro brincar um pouco no console.

```
vetor = [1, 2, 3]
println(vetor[1])
println(length(vetor))
vetor[2] = vetor[2] + 1
vetor[1] = 2 * vetor[3]
println(vetor)
```

Como disse antes, o for foi feito para manipular vetores, vamos ver umas funções, a primeira que imprime os elementos de um vetor um por linha.

```
function imprimeVetor(v)
  for el in v
    println(el)
  end
end
```

Isso também pode ser feito através dos índices do vetor:

```
function imprimeVetor(v)
  for i in 1:lenght(v)
    println(v[i])
  end
end
```

Como cada posição é independente, podemos calcular a soma dos elementos ímpares de um vetor

```
function somaImpVetor(v)
         soma = 0
         for i in 1:length(v)
           if v[i] % 2 == 1
             soma = soma + v[i]
           end
         end
         return soma
       end
```