Introdução à Programação e à R - Parte 1

Curso de Programação - 2025

João Pedro de Freitas Gomes

12 de Maio, 2023

Made
centro de
pesquisa em
macroeconomia
das desigualdades

Resumo

1. Principais Conceitos de Programação

2. R

3. Prática

Principais Conceitos de Programação

Por que utilizamos programação para resolver problemas?

Por que utilizamos programação para resolver problemas?

Utilizamos programação para a resolução de problemas porque não queremos fazer tarefas repetitivas e chatas. Ao quebrar qualquer tarefa em uma série de passos lógicos, podemos transformar esses passos lógicos em uma série de instruções que um computador deve seguir.

Algoritmos

- Um algoritmo é qualquer procedimento computacional bem definido que que toma algum valor ou conjunto de valores como entrada e produz algum valor ou conjunto de valores como saída.
- Em português claro, é justamente a sequência de passos lógicos comentados no slide anterior.

Estruturas de Decisão

- Imagine que você quebrou uma tarefa em uma série de passos lógicos (algoritmos), mas que um determinado passo dela pode ser diferente a depender do resultado de um passo anterior, como lidar com isso?
- Para isso, utilizamos estruturas de decisão!
- Elas basicamente mudam o passo a seguir a depender de algum resultado anterior.

Estruturas de Repetição

- Agora, imagine que algum passo de um algoritmo que você fez se repete várias e várias vezes, não faz muito sentido ter que repeti-la no código, correto?
- Para isso, existem as estruturas de repetição, que repetem um determinado trecho de código.

R

- R é uma linguagem de programação interpretada e multi-paradigma, no entanto, o paradigma mais relevante dela é o funcional.
- R vem de uma outra linguagem de programação chamada S. O nome “S” não é por acaso, ele vem de *Statistical*, justamente porque era uma linguagem voltada para a estatística. Ou seja, o R é uma linguagem que já nasce com o propósito de ser uma ferramenta fácil de se utilizar como um pacote estatístico.

Variáveis

Podemos criar variáveis de diversos tipos em R, seguem alguns exemplos:

```
n <- 15
nome <- "João"
vetor_de_numeros <- c(1,2,3,4,5)
lista <- list(nomes = c(Joao, Maria, José),
numeros = c(1,2,3,4,5), objeto = "computador")
```

Estruturas de Decisão

```
n <- 15
if(n==10){
    print("O número é 10)
}
```

Estruturas de Decisão

```r
n <- 15
if(n==10){
    print("O número é 10")
}
else{
    print("O número não é 10")
}
```

Estruturas de Decisão

```r
n <- 15
if(n==10){
    print("O número é 10")
}
else if(n<10){
    print("O número é menor que 10")
}
else if(n>10){
    print("O número é maior que 10")
}
```

Estruturas de Repetição

```
numeros <- 1:15
for(i in numeros){
    print(i)
}
```

Estruturas de Repetição

```
numeros <- 1:15
for(i in numeros){
    if(i %% 2 == 0){
        print(i)
    }
}
```

Estruturas de Repetição

```
numeros <- 1:15
numeros_2 <- c()
for(i in numeros){
    numeros_2 <- numeros[i]**2
}
```

Elementos Funcionais

- Como comentado anteriormente, um dos paradigmas mais relevantes para o R é o funcional, isto é, a ideia é que se resolva grande parte dos problemas propostos por meio de funções, evitando o uso de loops, por exemplo.
- Um dos elementos que caracterizam linguagens funcionais é a presença de **funções de primeira classe.** Ou seja, funções são uma estrutura de dados como qualquer outra.
- As funções da família *apply()* permitem escrever códigos em R em um estilo mais próximo do funcional.

Elementos Funcionais

```r
# Estilo imperativo
numeros <- 1:15
numeros_2 <- c()
for(i in numeros){
    numeros_2 <- numeros[i]**2
}
# Estilo funcional
quad <- function(x) x**2
numeros_2 <- lapply(numeros, quad)
```

Prática

Exercício

- Crie um *script* em R que crie duas listas: Uma que vai de 1 até 20 e outra que seja baseada nessa lista, mas os números pares estão elevados ao quadrado e os ímpares estão multiplicados por 2.