Capítulo

3 INICIANDO NO R

3.1 COMANDOS E AMBIENTE DE TRABALHO

Comandos no R são expressões inseridas no prompt ">" e finalizadas com a tecla Enter, é executado um comando. O prompt já apresentando automaticamente e indica que o R está pronto para receber um comando[1], se o prompt for digitado junto com o comando, uma mensagem de erro será emitida. Cada linha representa um comando, alternativamente pode-se inserir vários comandos em uma mesma linha, porém cada comando deve estar separado dos demais por ponto e vírgula "**;**".

Para executar o primeiro comando é necessário fazer o download e instalação do software R no Desktop. O download pode ser feito no site https://www.r-project.org/ e o usuário deve se atentar para escolher a alternativa de acordo com seu sistema operacional, uma vez que o R é multiplataforma,

[1]Além do prompt, outros símbolos podem ser apresentados, como "+" que indica que o comando anterior não foi finalizando, isso ocorre quando criamos blocos de comandos que ocupam mais de uma linha, como é o caso de funções. Também pode ocorrer quando equivocadamente esquecemos de fechar um parêntese e o comando ainda está aguardando, neste caso pode-se fechar o parêntese na próxima linha, se possível, caso contrário pode-se usar a tecla Esc para cancelar o comando.

versões para Windows, Mac e diversas distribuições Linux são disponibilizadas.

Após a instalação, o R pode ser inicializado nas versões 32 bits (R i386) ou 64 bits (R x64). Atualmente a maioria dos processadores e sistemas operacionais são de 64 bits, então preferencialmente opte pela versão de 64 bits, até mesmo porque na versão de 32 bits existe um limite teórico de endereçamento de 2^{32} = 4.294.967.295 = 4 Gb na memória RAM do computador, que na prática pode ser considerado de 2 Gb. E assim, objetos superiores a 2 Gb não poderão ser trabalhados diretamente na memória RAM do computador. Pode parecer muito, mas 2 Gb podem ser rapidamente consumidos em operações relativamente simples, conforme o tamanho da base de dados original. Isto é uma realidade, sobretudo em análises espaciais, em que além da característica (atributos), coordenadas também devem ser armazenadas.

Após inicializado, você já pode adicionar seu primeiro comando:

```
> 9 + 2
[1] 11
```

O comando acima executa uma simples soma de 9 e 2, o comando é precedido prompt ">" e o resultado é apresentado na linha seguinte, precedido do número 1 entre colchetes. O número 1 representa o índice do resultado, ou seja, a posição do elemento no vetor. Os números índices e colchetes apresentados são meramente ilustrativos, o devido tratamento a índices e indexação será dado no capítulo sobre vetores e

demais estruturas de dados do R. Na tela do software, mais em destaque o console (ou R console) será apresentado algo semelhante à Figura 1.

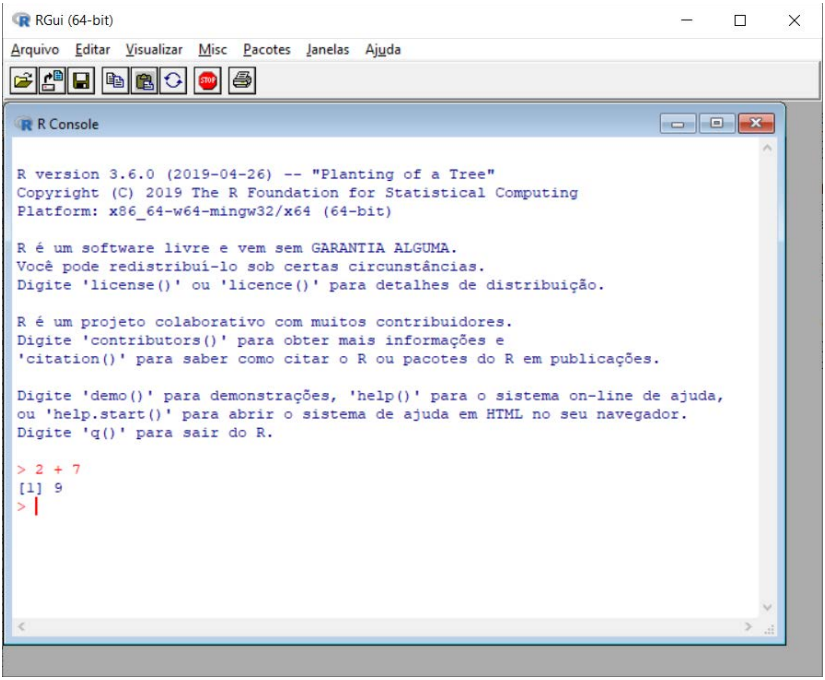

Figura 1. Ambiente de trabalho do R.

O console é a parte do *software* que efetivamente executa as operações, alguns usuários trabalham diretamene nele, em um processo de formular → digitar → executar os procedimentos. Porém, a medita que esses procedimentos adquirem o mínimo de complexidade, trabalhar diretamente no console torna-se improdutivo. Então com o auxílio de um editor de textos (como o blocos de notas ou o próprio editor do R, acessado via Arquivo

- Abrir script), o usuário trabalha em um processo cíclico de formular ↔ digitar, e então o procedimento é executado (enviado/copiado para o console) após a finalização parcial ou final deste ciclo.

Apesar do R disponibilizar um editor de texto específico para editar scripts trata-se de um editor bastante limitado, e assim a ampla maioria dos usuários optam por instalar um segundo software para auxiliar na tarefa de edição de scripts. No caso da linguagem de programação R a maioria esmagadora dos usuários optam pelo RStudio, que pode ser considerado um Ambiente de Desenvolvimento Integrado - Integrated Development Environment (IDE).

O RStudio apresenta caracteristicas apreciáveis, tais como, plataforma madura, amplamente utilizada e com um excelente editor de texto, que conta com uma série de funcionalidades, como identificação de erros de sintaxe; complemento de funções e objetos; coloração diferenciada de objetos e estruturas de controle; atalhos de teclado úteis; além de outras funcionalidades, como janelas específicas para plotar figuras, acessar arquivos e bases de dados, consultar os documentos de ajuda das funções (Figura 2).

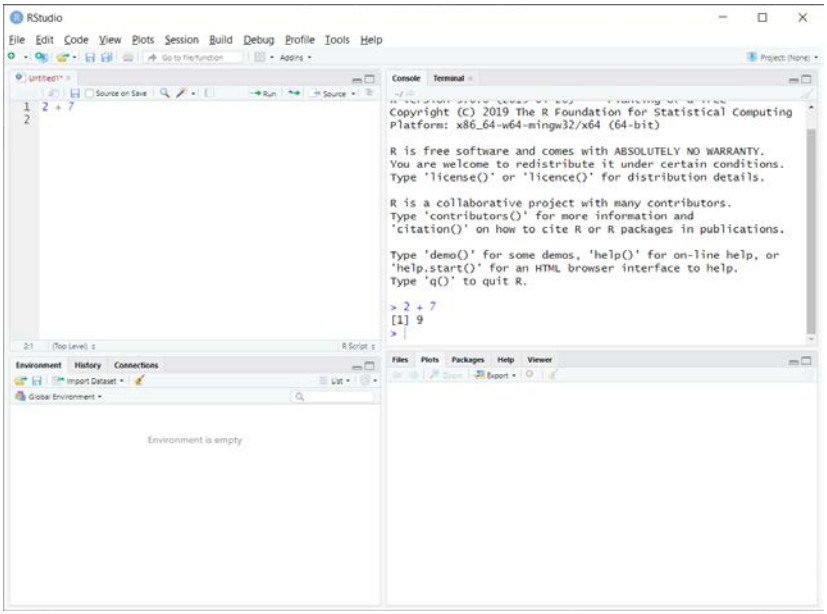

Figura 2. Ambiente de trabalho do R acessado via RStudio.

O download do RStudio pode ser feito no site https://rstudio.com/products/rstudio/download/. O RStudio não substitui o R, ele é apenas uma interface mais agradável e produtiva para acessar todos os procedimentos poderosos que o R oferece, então antes de instala-lo deve-se instalar o R.

A seguir é a apresentado mais algumas operações matemáticas básicas, a partir de agora com a omissão do prompt e adição de comentários nos resultados retornados pelo programa[2].

[2] Essas medidas serão particularmente úteis para facilitar operações de copiar-colar códigos da versão digital deste documento.

```
2 + 3*4 # prioridade da operação multiplicação
## [1] 14
3/2 + 1 # prioridade da operação divisão
## [1] 2.5
3 / (2 + 1) # parênteses estabelecem prioridade
## [1] 1
2 * 3 ^ 2 # potências são indicadas por ^ ou **
## [1] 18
```

Determinados operadores apresentam prioridades sobre os demais, assim como qualquer em qualquer calculadora ou planilha eletrônica. Espaços entre os operados podem ser reservados ou não entre os números, a legibilidade dita a quantidade de espaços reservados. Linhas que apresentam o símbolo "#" definem um comentário e são ignoradas pelo R.

No R todas as funções têm a seguinte forma:

função(argumento(s)obrigatório(s), argumento(s)opcional(is))

Sendo que os argumentos opcionais podem ter um valor padrão pré-estabelecido ou não. Os argumentos estarão sempre entre parênteses sendo separados por vírgula.

```
log(2) # ln - Logaritmo de 2 na base e
## [1] 0.6931472
log(2, 10) # Logaritmo de 2 na base 10
## [1] 0.30103
```

Se você deixar o primeiro argumento (obrigatório) em branco, vai receber uma mensagem de erro:

```
log( , 3)
## Error: argument "x" is missing, with no default
```

A seguir, apresentamos uma lista de algumas funções básicas e operadores aritméticos:

Tabela 1: Lista de funções e operadores aritméticos.

Função	Descrição
sqrt( )	raiz quadrada
abs( )	valor absoluto
exp( )	exponencial de base “e”
log10( )	logaritmo na base 10
log( )	logaritmo na base “e” ou LN
sin() cos() tan()	funções trigonométricas
asin() acos( ) atan( )	funções trigonométricas inversas
+ - * / ** ou ^	adição, subtração, multiplicação, divisão e potência

Para pedir ajuda ao R e ter acesso à documentação de determinada função, por exemplo, para a função log, digite uma das opções:

```
help(log)
?log
```

Pesquisamos a documentação oficial do programa com muita frequência, praticamente o tempo todo, pelo menos para saber quais são os argumentos necessários para executar determinada função. Uma das opções acima é para situações em que você já sabe o nome do comando, porém sempre você pode utilizar o mecanismo de completar funções do RStudio ou do próprio console do R para identificar nomes de funções.

Para pesquisar sobre temas em particular utilize os comandos abaixo. Porém na maioria das vezes é mais interessante fazer essa pesquisa mais abrangente no *Google*.

```
??logarithms
help.search("logarithms")
```

Outra função útil de ajuda no R é a função **args**, que lista os argumentos necessários para executar a função de interesse, porém a maioria dos usuários prefere acessar a documentação completa da função que além de listar também descreve os argumentos e cada um dos elementos que compõe a função.

```
args(log)
## function (x, base = exp(1))
##   NULL
```

3.2 GERANDO E REMOVENDO OBJETOS

As entidades nas quais R opera são tecnicamente conhecidas como objetos. Exemplos são vetores de valores numéricos (reais) ou complexos, vetores de valores lógicos e vetores de cadeias de caracteres.

O R é uma linguagem orientada a objetos. Um objeto para o R significa tanto um banco de dados, como uma tabela, variáveis, vetores, matrizes, funções, etc., armazenados na memória ativa do computador. Para criar um objeto qualquer no R, você deverá sempre usar o operador de atribuição "<-", gerado pela digitação do sinal de menor e menos.

```
x <- sqrt(9)
```

Pronuncia-se o comando dizendo: o objeto recebe certo valor. Por exemplo, `x <- sqrt(9)`, leia-se, "x recebe a raiz quadrada de 9".O objeto x, armazenou a raiz quadrada de 9. Verifique, digitando x:

```
x
## 3
```

Existem várias formas de fazer atribuições de objetos além do operador "<-". Outras três delas são apresentas abaixo.

```
sqrt(9) -> x; x = sqrt(9); assign("x", sqrt(9))
```

Porém convencionalmente os usuários do R adotam o "<-", então é altamente recomendado que este deve ser adotado,

pois com o costume você se sentirá mais confortável com códigos de terceiros e o oposto também é verdadeiro.

Ao se fazer uma atribuição deve-se atentar para o fato de que um objeto substitui o outro de mesmo nome, e nenhuma mensagem de advertência é emitida. Para avaliar o conteúdo do objeto, isto é, imprimir o conteúdo na tela, basta digitar o nome do objeto e pressionar a tecla Enter. Basicamente, esse comando chama internamente a função de impressão `print`.

```
obj1 <- 25
print(obj1)
## [1] 25
obj1
## [1] 25
obj1 <- 21 + 1
obj1
## [1] 22
```

Na maioria das vezes utilizamos a forma resumida do comando `print`, mas em algumas situações seu uso é obrigatório, como dentro de um processo repetitivo, que será apresentado mais adiante neste livro.

Outra função útil de impressão de objetos é a função `cat` (concatenar e imprimir). Ela serve para concatenar (juntar) objetos e imprimi-los na tela ou até mesmo salva-los em um arquivo, isso mesmo, a impressão dos objetos pode ser direcionada para um arquivo texto. Essa função é muito utilizada para concatenar um texto com resultados derivados da

execução de algoritmos, armazenados em objetos. A `cat` é mais flexível e "personalizável" do que a função `print`.

```
cat("O valor do objeto 1 (obj1) é", obj1)
## O valor do objeto 1 (obj1) é 22
```

O comando `print` combinado com a função `paste` pode gerar resultado semelhantes à função `cat`. A função `paste` serve para concatenar (juntar) vetores após converte-los em caracteres (texto).

```
print(paste("O valor do objeto 1 (obj1) é", obj1))
## [1] "O valor do objeto 1 (obj1) é 22"
```

O R reconhece letras maiúsculas e minúsculas como caracteres diferentes, assim como a ampla maioria das linguagens de programação, essa característica recebe o nome de *case sensitive*. Observe o comportamento dos objetos abaixo.

```
a <- 1; A <- 5
nome <- "Eduardo"
Nome <- "outro nome"
a; A; nome; Nome
## [1] 1
## [1] 5
## [1] "Eduardo"
## [1] "outro nome"
```

Durante uma sessão do R, os objetos são criados e guardados por nomes. Para saber quais objetos estão guardados na memória pelo R basta avaliar a aba de Environment do RStudio ou utilizar um dos comandos:

```
objects()
## [1] "a"    "A"    "nome" "Nome" "obj1" "x"
ls()
## [1] "a"    "A"    "nome" "Nome" "obj1" "x"
```

Para eliminar um ou mais objetos basta utilizar a função `rm` de remover.

```
rm(x, obj1)
ls()
## [1] "a"    "A"    "nome" "Nome"
```

Para eliminar todos os objetos pode-se utilizar o comando o comando abaixo, porém a maioria dos usuários preferem utilizar a opção disponível no menu do programa para essa funcionalidade, em Misc – Remover todos os objetos, no R padrão e em Session – Clear Workspace, no RStudio.

```
rm(list=ls(all=TRUE))
```

Se o seu interesse é apenas limpar o console, pode-se utilizar o atalho ctrl+L, porém esse atalho não remove os objetos.

É comum errar alguns comandos quando se está trabalhando com o R. Seja pela falta de familiaridade com o comando ou então por algum erro de digitação. Para evitar ter que escrever o comando todo de novo, utilize a seta do teclado de direção para cima para pesquisar todo histórico de comandos utilizados na sessão atual. Quando encontrar o comando desejado faça as devidas correções e execute novamente. Porém é altamente recomendado que o processo de criação de um procedimento seja feito utilizando o editor de texto, assim a preocupação de

recuperar comandos é eliminada, uma vez todos os comandos estarão salvos no arquivo texto de edição. E no caso de um eventual erro, basta fazer a correção no arquivo texto e executar o procedimento novamente no console.

Capítulo

4 CRIAÇÃO DE FUNÇÕES

No tópico anterior foi mostrado como executar uma função no R, neste tópico iremos mais além, será apresentado os componentes básicos para você criar suas próprias funções e se beneficiar de todas as suas vantagens.

Uma das maiores potencialidades do R é que permite ao usuário definir suas próprias funções de forma simples e fácil. Isso o torna uma ferramenta poderosa para criar e testar novas metodologias. As funções são utilizadas para praticamente tudo e inclusive para criar novas funções. No R as funções apresentam papel de destaque, pois é a principal forma de interagir com as rotinas nativas da linguagem.

4.1 MOTIVAÇÃO PARA CRIAR FUNÇÕES

De uma visão mais prática as funções são úteis por uma série de fatores, podemos citar estender/expandir as funcionalidades de um sistema base, isto é, criar novos procedimentos para o sistema; serve para encapsular parte do código que executa a mesma funcionalidade ao longo do processo, e assim organizar melhor o código, além de tornar o processo mais seguro, uma vez que a execução da função ocorre em ambiente local (diferente do ambiente global da sessão), isso pode evitar efeitos indesejáveis, os processos do ambiente local não afeta o global e também não é afetado; permite reutilizar códigos, e

assim evitar repetição desnecessária de códigos, isso facilita muito o processo de criação e manutenção do código, além de evitar o problema de copiar e colar: as funções são extremamente úteis para reduzir a duplicação de códigos[BOX1].

BOX1: Métrica do código repetido
Existem softwares, tais como CodeClimate e SonarQube, que avaliam a qualidade do código desenvolvido frente a diferentes perspectivas, uma delas é a quantidade de códigos repetidos. Códigos repetidos são nocivos para o programa, uma vez que o torna mais extenso e complexo. Em caso de necessidade de alteração no código repetido a alteração deverá ser feita em todos as repetições, sem exceções, e isso pode ser trabalhoso e com alta susceptibilidade à erros. O ideal seria encapsular este trecho de código repetido em uma função.

De uma visão mais geral, o papel primordial das funções é o de abstração. Abstração é difícil de ser definida, mas consiste no ato de isolar elementos em detrimento a outros. A abstração é muito útil para simplificar eventos e assim facilitar a resolução de problemas, por exemplo, isola-se apenas aquilo que é útil e relevante, todo resto então deve ser esquecido ou eliminado.

Um exemplo prático de abstração seria o uso de um projetor de slides. Um professor não precisará saber que tipo de lâmpada aquele projetor utiliza, nem mesmo a quantidade de lumens da lâmpada para utilizá-lo em uma aula. Muito provavelmente o conhecimento necessário será o de instalar em um computador e a funcionalidade de liga/desliga. Apenas com esse

conhecimento, o professor já é capaz de ministrar sua aula. Observe que um nível de abstração diferente é requerido para um técnico que ganha a vida dando manutenção nesses equipamentos, possivelmente ele vai ter que entender de lâmpadas e lumens.

Na programação, um exemplo poderia ser a função `nnet` da biblioteca de mesmo nome do R, essa função serve para ajustar redes neurais artificiais à um conjunto de dados. A interface com o usuário é simples e intuitiva, indicada para a maioria das pessoas, que irão utilizar essa função como uma ferramenta, como um Engenheiro Florestal que irá predizer a produtividade de determinada cultura utilizando variáveis climáticas. Um nível de abstração diferente será requerido para um profissional da matemática ou ciência da computação, que será responsável por implementar uma função que executa uma rede neural artificial.

4.2 ELEMENTOS BÁSICOS DE UMA FUNÇÃO

A nova função R que você construir poderá ser completamente nova (um novo modelo que você está testando, por exemplo) ou apenas uma modificação personalizada de uma função do R já existente. Você pode desejar ainda usar as funções já existentes de modo repetido no seu conjunto de dados, isto facilitará em muito o seu trabalho, já que as tarefas a serem realizadas ficarão incorporadas em uma única função.

A sintaxe[BOX2] básica de uma função do R é apresentada a seguir.

Sintaxe:

```
nome <- function(argumentos){
      sequências de instruções (corpo)
      return(argumento)
}
```

nome: aquele que você escolhe para dar a função. O nome é um elemento essencial de uma função, para existir a função deverá possuir um nome;

argumentos: lista de expressões a serem usadas dentro da função, que podem ser obrigatórios ou opcionais. Os argumentos também podem apresentar um padrão pré-estabelecido. Argumentos não são essenciais, embora incomum, funções sem argumentos podem ser construída;

corpo: é a parte da função que realmente trabalha, é constituído por expressões R que serão avaliadas sequencialmente quando executadas. Não faz sentido construir uma função sem corpo, isto é, que não executa qualquer instrução, porém é possível construir uma função sem corpo e nenhum erro de sintaxe seria emitido.

return: é o último valor calculado, e pode vir acompanhado para função reservada `return` ou não. O retorno de uma função não é essencial, podemos ter interesse em apenas imprimir o resultado na tela. O retorno é útil se este resultado será utilizado posteriormente no decorrer no processo, assim o retorno

poderá ser armazenado em um objeto após a execução da função.

BOX 2: Sintaxe, semântica, lógica.

A **sintaxe** refere-se às regras que ditam a composição de textos com significado lógico para determinada linguagem programação. Os erros de sintaxe são erros no código e bloqueiam a execução de um processo. Exemplos seria digitar de forma errada a palavra reservada `function` ou esquecer de fechar um parêntese. O RStudio possui mecanismos para checar alguns erros sintaxe na edição do script, antes mesmo da execução. Os erros de sintaxe geralmente são fáceis de serem detectados. Além dos erros de sintaxe ainda podemos ter os erros de **semântica**, que envolvem códigos tecnicamente corretos, mas apresentam problemas com o significado. Esses erros geralmente só podem ser identificados no momento da execução do código. Um exemplo seria tentar ler um arquivo que não existe no computador. Há também os erros de **lógica**, esses erros são mais difíceis de serem detectados, a sintaxe e semântica estão corretas, mas mesmo assim o código não executa da maneira com que o programador imaginou. Os erros de lógica são críticos porque não podem ser detectados pelo editor ou interpretador, a identificação seria de responsabilidade inteiramente do programador. Exemplo seria usar uma operação errada no processo, como utilizar soma em vez multiplicação. O resultado seria gerado corretamente, porém é de nosso interesse a multiplicação e não a soma.

Podemos criar nossa própria função para dar boas-vindas ao usuário, observa as diferentes versões para a tarefa.

```
# Versão 1 – sem argumentos
boas_vindas <- function(){ # criar a função
  cat("Seja bem vindo!")
}
boas_vindas() # executar a função
## Seja bem vindo!

# Versão 2 – com argumentos
boas_vindas <- function(nome){ # criar a função
  cat("Seja bem vindo", nome, "!")
}
boas_vindas("Eduardo") # executar a função
## Seja bem vindo Eduardo !

# Versão 3 – com argumento padrão
boas_vindas <- function(nome = "Aluno"){ # criar a função
  cat("Seja bem vindo", nome, "!")
}
boas_vindas() # executar a função
## Seja bem vindo Aluno !

# Versão 4 – com argumento padrão e retorno
boas_vindas <- function(nome = "Aluno"){ # criar a função
  frase <- paste("Bom dia", nome, "!")
  return(frase)
}
frase_boas_vindas <- boas_vindas("Eduardo") # executar a função
frase_boas_vindas # mostrar o objeto resultante
## [1] "Bom dia Eduardo !"
```

O próximo exemplo demostra o uso do resultado retornado pela execução da função em uma análise posterior. Além de retornar

o resultado numérico, a função também imprime na tela o resultado combinado com um texto explicativo.

```
lucro <- function(receita, custo){
  valor_lucro <- receita - custo
  cat("O lucro foi de R$", valor_lucro, "\n")
  return(valor_lucro)
}
lucro_projeto1 <- lucro(6000, 4500)
## O lucro foi de R$ 1500
lucro_projeto2 <- lucro(4000, 5000)
## O lucro foi de R$ -1000
lucro_total <- lucro_projeto1 + lucro_projeto2
lucro_total
## [1] 500
```

A função computa o lucro de dois diferentes projetos, os resultados são somados para computar qual seria o lucro total. Observe que foi incluído o texto `"\n"` na função de concatenar e imprimir, isso foi necessário para evitar que a impressão e o resultado retornado sejam apresentados na mesma linha. O texto `"\n"` indica para o programa pular de linha. Vale destacar que se optar por utilizar a função print combinada com a função paste não seria necessário incluir `"\n"`, pois por padrão a função `print` já pula para próxima linha após a execução.

4.3 Escopo

Regras de espoco, ou simplesmente escopo, é um conjunto de regras destinado a controlar o acesso de variáveis durante a execução de um processo. Não existe um padrão universal de

regras de escopo, mas em geral, não existe grandes variações de regras entre a maioria das linguagens de programação.

Em se tratando de funções, no R temos dois escopos básicos de variáveis, o global e o local. Qualquer objeto definido

então cada linguagem de programação pode apresentar regras específicas de escopo, porém

O espoco de uma função está relacionado com seu ambiente de trabalho. Em se tratando de ambiente é necessário saber que existe dois ambientes básicos de execução, o ambiente o local e o global.