

Trabalhando com scripts

Scripts são os arquivos contendo os comandos que formam os programas em R. Esses arquivos devem ser gerados em editores de texto “puro”.

Exemplos de editores de texto:

Em Windows:

- Bloco de notas;
- Notepad++;
- Visual Studio Code

Em Linux:

- Vim;
- Emacs;
- Nano;
- Gedit

Também é possível gerar os scripts pelo editor de script do R. Esse editor pode ser acessado pelo menu:

Arquivo – Novo Script ou Arquivo – Abrir Script

Dentro de um script R além dos comandos, podemos ter comentários. Comentários são essenciais para a documentação do código, além de explicar trechos do código. Os comentários não são lidos pelo interpretador. Para inserir um comentário basta colocar o sinal “#” no início do linha.

Os scripts R podem ser salvo em formato texto “txt”, porém é uma boa prática salvar esses arquivos com a extensão “.r” isso deixa seu ambiente mais organizado.

Para executar o script no R, podemos usar as opções:

Menu arquivo – Interpretar código fonte R ou
source(“caminho e nome do arquivo”)

Entrada de dados pelo usuário e exibição de respostas.

Ao se criar um programa em R, muitas vezes precisamos que o programa interaja com os usuário solicitando informações. Embora a linguagem R seja voltada mais aos cálculos e não para programas interativos, em alguns casos o usuário vai ter que confirmar informações, inserir dados ou solicitar o fim do programa.

Para solicitar a entrada de dados pelo usuário, utilizamos a função readline. Sintaxe:

readline(prompt=”Texto”)

O parâmetro prompt é o texto que será exibido ao usuário. Veja o exemplo:

readline(prompt=”Digite sua idade”)

Para receber o dado digitado, utiliza-se uma variável:

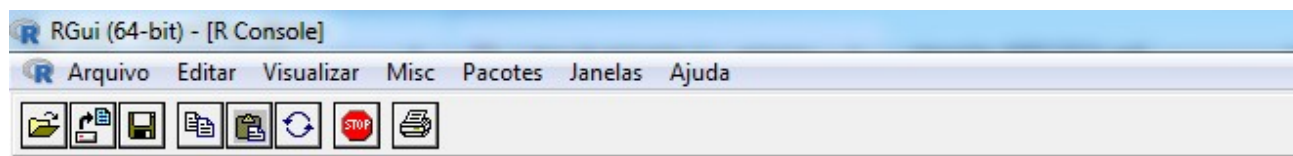
```
idade <- readline(prompt="Digite sua idade")
```

A variável vai receber o dado em formato texto mesmo que seja um número. Se esse dado for ser utilizado em cálculos deve-se realizar a conversão com a função `as.numeric`. Veja:

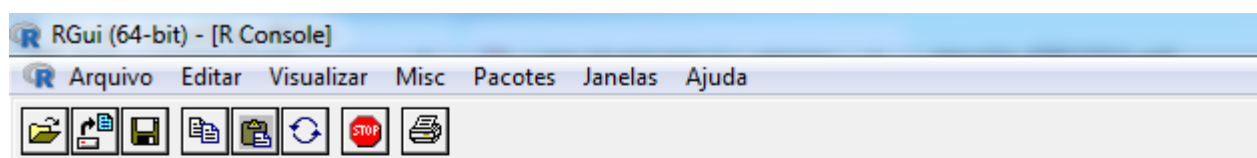
```
dado <- readline(prompt="Digite sua idade")
ano_atual <- 2023
idade <- as.numeric(dado)
ano_nascimento <- ano_atual - idade
```

No exemplo acima, calculamos o ano de nascimento baseado na idade informada pelo usuário, como a idade vem na forma de texto, convertemos com a função `as.numeric`. Para exibir informações ao usuário, podemos utilizar as funções `print` ou `cat`. A função `cat` permite caracteres de escape. Os caracteres de escape não são exibidos, tendo como função colocar o cursor na próxima linha, no caso do caracter `"\n"` ou dar um `"tab"` no caso do caracter `"\t"`.

Veja as telas com o uso das funções:



```
> idade <- 23
> ano_atual <- 2023
> ano_nascimento <- ano_atual - idade
> print(ano_nascimento)
[1] 2000
> |
```



```
> texto <- "1ª Linha \n Segunda linha"
> print(texto)
[1] "1ª Linha \n Segunda linha"
> cat(texto)
1ª Linha
Segunda linha> |
```

Operadores Aritméticos

Além das operações tradicionais:

/ Divisão

* Multiplicação

- Subtração

+ Adição

Na linguagem R temos também os operadores:

%% - Esse operador retorna o resto da divisão entre dois números;

%/% - Esse operador retorna a parte inteira da divisão

Quando efetuamos uma divisão com o operador "/" podemos ter como resultado valores decimais no caso de um resultado não-inteiro. Como exemplo a divisão 13/4 vai gerar o resultado 3.25. Para separarmos a parte inteira do resto usamos os operadores acima. Assim:

13%%4 = 1 (resto da divisão)

13%/%4 = 3 (parte inteira de 3.25)

Vetores

Vetores são variáveis que armazenam vários valores, diferente das variáveis tradicionais que suportam apenas um valor. Cada valor fica endereçado por um índice na variável. Para criar um vetor em R, usamos a função c. Veja o exemplo:

```
numeros <- c(10,20,30,40,50)
```

Cada número vai ficar endereçado da seguinte forma:

índice	1	2	3	4	5
valor	10	20	30	40	50

Para referenciar valores individuais do vetor basta chamar nome e índice. Assim:

```
numeros[1] = 10
```

E assim por diante.

Laços de repetição

Quando trabalhamos com vetores, é muito útil utilizar laços de repetição para manipular o vetor.

Imagine o exemplo.

Temos os vetores:

```
nomes <- c("José","Maria","Pedro")
```

```
salarios <- c(5000,10000,4000)
```

Vamos exibir uma lista com nome seguido do salário.

Podemos usar o laço for para isso

O laço for tem a seguinte sintaxe:

```
for(escopo)
  comandos
```

O escopo é o espaço de valores dentro dos quais os comandos serão executados.

No exemplo acima, precisamos executar o laço três vezes.

O comando fica assim:

```
for (i in 1:3){
  cat("Nome: ",nomes[i],"Salário: ",salarios[i],"\\n")
}
```

Veja a saída abaixo:

Nome: José Salário: 5000

Nome: Maria Salário: 10000

Nome: Pedro Salário: 4000

Comando While

A tradução desta função nos ajuda a entendê-la mais: while significa enquanto. Então podemos ler essa função como: Enquanto alguma condição for verdadeira, o código abaixo será repetido.

Sintaxe:

```
while (condição){
  comandos a serem repetidos
}
```

Exemplo:

a=1

```
while (a <=10 ) {
```

```
  print (a)
```

```
  a=a+1
```

```
}
```

Note que os dois últimos exemplos resultam na mesma coisa: o R vai retornar os números de 1 a 10 em sequência. Porém nós precisamos mudar o valor de a para que a sequência continue no caso do while () enquanto no for () a sequência progride sem precisarmos fazer isso manualmente. Além disso, ao usar while () precisamos declarar a variável antes para que o R possa testar a condição expressa dentro dos parênteses.

O comando if e os operadores de comparação e lógicos

Em toda linguagem de programação, testar condições e executar comandos baseados na verdade ou não dessas condições é essencial.

Em R para fazer isso usamos o comando if:

```
if(condição){  
  comandos que serão executados se a condição for verdadeira  
} else{  
  comandos que serão executados se a condição for falsa  
}
```

Na condição temos de fazer testes de comparação ou lógicos. Veja os exemplos:

Os operadores de comparação em R são os seguintes:

Operador	Significado
==	igual a
!=	diferente de
>	maior que
<	menor que
>=	maior ou igual a
<=	menor ou igual a

Os operadores lógicos disponíveis em R são:

Operador	Descrição	Exemplo	Explicação
&	AND lógico	<pre>> x <- 0:2 > y <- 2:0 > (x < 1) & (y > 1) [1] TRUE FALSE FALSE</pre>	Versão vetorizada. Compara dois elementos do tipo vetor e retorna um vetor de TRUES e FALSEs
&&	AND lógico	<pre>> x <- 0:2 > y <- 2:0 > (x < 1) && (y > 1) [1] TRUE</pre>	Versão não-vetorizada. Compara apenas o primeiro valor de cada vetor, retornando um valor lógico.
	OR lógico	<pre>> x <- 0:2 > y <- 2:0 > (x < 1) (y > 1) [1] TRUE FALSE FALSE</pre>	Versão vetorizada. Compara dois elementos do tipo vetor e retorna um vetor de TRUES e FALSEs
	OR lógico	<pre>> x <- 0:2 > y <- 2:0 > (x < 1) (y > 1) [1] TRUE</pre>	Versão não-vetorizada. Compara apenas o primeiro valor de cada vetor, retornando um valor lógico.
!	NOT lógico	<pre>> x <- 0:2 > y <- 2:0</pre>	Negação lógica. Retorna um valor lógico único ou um vetor de TRUE / FALSE.

Operador	Descrição	Exemplo	Explicação
		<pre>> !y == x [1] TRUE FALSE TRUE</pre>	
xor	XOR	<pre>> x <- TRUE > y <- FALSE > xor(x,y) [1] TRUE</pre>	<p>Ou Exclusivo. Retorna valor lógico TRUE se ambos os valores de entrada forem diferentes entre si, e retorna FALSE se os valores forem iguais.</p>