

UNIVERSIDADE DA REGIÃO DE JOINVILLE - UNIVILLE

Bacharelado em Engenharia de Software (BES)

Estatística para computação

Professora Priscila Ferraz Franczak

Engenheira Ambiental - UNIVILLE

Mestre em Ciência e Engenharia de Materiais - UDESC

Doutora em Ciência e Engenharia de Materiais - UDESC

priscila.franczak@gmail.com

Plano de Aula

- 1. Entendendo o conceito de função**
- 2. Funções envolvendo Loops**
- 3. Funções envolvendo Condição**
- 4. Operadores Lógicos**
- 5. Funções matemáticas**
- 6. Exercícios**

1. Entendendo o conceito de função

- Uma das maiores vantagens do R é a facilidade de criar novos comandos (funções).
- Porém já existem diversos comandos disponíveis nos pacotes do R.
- Apesar da facilidade em criar funções, não se deve descartar a possibilidade de elas já terem sido criadas.

- Para criar uma função, geralmente são necessárias as seguintes atribuições:

```
nomefuncao<-function(argumento1,...,argumento)
```

- Depois disso, digita-se o conteúdo da função, usando chaves { } para dizer ao R onde o conteúdo começa e termina.

Exemplo:

Vamos começar com uma função que subtrai dois objetos. É programada como um algoritmo, passo a passo).

```
> menos<-function(a,b)  #criando função
+ {                      #início
+   result<-a-b          #subtraindo
+   return(result)       #valor a ser retornado
+ }                      #fim
> menos(1,2)
[1] -1
```



```
> menos (a=4 , b=2)
```

```
[1] 2
```

```
>
```

```
> menos (4 , 2)
```

```
[1] 2
```

```
>
```

```
> menos (b=2 , a=4)
```

```
[1] 2
```

```
>
```

```
> menos (2 , 4)
```

```
[1] -2
```

- O R já possui o comando de subtração de elementos (-), mas criamos a função batizada de menos().
- As novas funções nada mais são que objetos do R da classe `function`.
- Importante não criar um objeto com um nome já existente, pois o novo objeto apagará o antigo.

- Uma forma fácil de verificar se um nome que você pretende usar já está sendo usado por algum objeto, é digitar o nome no console do R:

```
> teste
```

```
Error: object 'teste' not found
```


- O objeto retornado da função `menos()` é o objeto `result`.
- Nos exemplos o resultado foi apenas exibido, sem armazená-lo.
- Para armazenar o resultado de uma função criada, procede-se exatamente como para os comandos já existentes.

```
> x<-menos(4,2)
```

```
> x
```

```
[1] 2
```

- Uma função também pode ser programada para exibir algo na tela:

```
> teste<-function(obj)
+   print(obj)
> teste(obj=1:3)
[1] 1 2 3
> teste      #visualizando o conteúdo da função teste
function(obj)
  print(obj)
```


Exemplo:


Jogada de moedas (cara ou coroa):

- Neste caso a função terá dois argumentos (x e n).
- **x** será a “moeda” c(“cara”, “coroa”) e **n** será o número de vezes que deseja jogar a moeda.

```
> jogar.moeda<-function(x,n){  
+   sample(x,n,replace=T)  
+ }  
>  
> moeda<-c("Cara", "Coroa")  
> jogar.moeda(moeda,2)  
[1] "Coroa" "Coroa"  
> jogar.moeda(moeda,10)  
[1] "Coroa" "Cara" "Cara" "Cara" "Cara" "Cara"  
[7] "Coroa" "Cara" "Coroa" "Cara"
```

Podemos observar que:

- Como a função executa apenas uma linha de comando `print(obj)` não foi necessário o uso das chaves.
- Uma vez que não usamos o comando `return()` dentro da função `teste()`, ela retorna o valor da última expressão executada, ou seja, `print(obj)`.

- 
- Para que uma função exiba o conteúdo de um objeto na tela não basta apenas digitar o nome do objeto, é preciso declará-lo explicitamente, por exemplo, no comando `print()`.
 - Para visualizar o conteúdo de uma função, basta digitar o nome no console.

2. Funções envolvendo Loops

- **Loop** é um processo iterativo (que se repete várias vezes), cujo objetivo é a realização de uma sequência de comandos até uma condição previamente estabelecida.
- As iterações devem ter condições finitas.

- Pode-se utilizar comandos como `while` (condição) ou `for` (condição).
- Para o `for`, usamos um contador.
- O termo `(i in 1:10)` quer dizer que o contador, que chamamos de `i`, assumirá valores 1,2,3,4,5,6,7,8,9,10 (sempre incrementando inteiros).

Exemplo:

- Função usando o for:

```
> loop1<-function(objeto)
+   for(i in 1:3) print(objeto)
> loop1("teste")
[1] "teste"
[1] "teste"
[1] "teste"
```


- Mesma tarefa anterior, usando while:

```
> loop2<-function(objeto)
+ {
+   i<-1
+   while(i<4)
+   {
+     print(objeto)
+     i<-i+1
+   }
+ }
> loop2("teste")
[1] "teste"
[1] "teste"
[1] "teste"
.
```

Criando um contador

Condição do while

i vai crescendo até...

3. Funções envolvendo Condição

- No R as funções envolvendo condições são programadas com if e else.

```
> x<-1  
> if(x==1) print("sim") else print("não")  
[1] "sim"
```

- Alternativamente, pode-se usar o comando `ifelse()`.

```
> x<-1  
> ifelse(x==1,print("sim"),print("não"))  
[1] "sim"
```


Exemplo:

- Crie um objeto com o valor do salário de dez pessoas, com os valores: 1000,400,1200,3500,380,3000,855,700,1500,500.
- Consideramos que as pessoas **que ganham menos de 1000 ganham “pouco”**. Então aplicamos o teste e pedimos para retornar “pouco” para quem ganha menos de 1000 e **“muito” para quem ganha mais de 1000**.

```
> salarios<-c(1000,400,1200,3500,380,3000,855,700,1500,500)
> ifelse(salarios<1000,"pouco","muito")
[1] "muito" "pouco" "muito" "muito" "pouco" "muito"
[7] "pouco" "pouco" "muito" "pouco"
```

- Pode-se usar apenas o comando `if()`.

```
> if(x==1) print("sim")  
[1] "sim"
```


- Os comandos de condição podem ser usados dentro de funções:

```
> cond1<-function(obj)
+   if(obj==1) print(obj)
> cond1(1)
[1] 1
```

- Se o objeto for 2, não imprime valor.

4. Operadores lógicos

- Além dos operadores lógicos:

`==, !=, <, >, <=, >=`

Temos ainda:

Símbolo	Significado
!	Lógico de negação – NÃO
&	Lógico E
	Lógico OU
&&	E com SE*
	OU com SE**

*se a primeira expressão for FALSE, o R nem testa a segunda.

**se a primeira expressão for TRUE, ele nem testa a segunda.

• Exemplos

```
> 1==1 #1 é igual a 1?
[1] TRUE
> (1==1)&(2==2) #1 é igual a 1 E 2 é igual a 2?
[1] TRUE
> (1!=1)&(2==2) #com & o R testa as duas condições
[1] FALSE
> (1!=1)&&(2=2) #como a primeira é falsa, ele nem testa a segunda
[1] FALSE
> (1==1)||(1==2) #como a primeira é verdadeira, ele nem testa a segunda
[1] TRUE
.
```


5. Funções matemáticas

- Combinatória:

Calcula o número de combinações de n elementos em grupos de tamanho k :

`choose(n, k)`

```
>choose(3, 2)  
[1] 3
```

- Fatorial

Calcula o fatorial de x:

`factorial(x)`

```
>factorial(3)
```

```
[1] 6
```

```
>factorial(6)
```

```
[1] 720
```

- Raiz quadrada

Calcula a raiz quadrada de x:

`sqrt(x)`

```
> sqrt(49)
```

```
[1] 7
```

```
> sqrt(36)
```

```
[1] 6
```




6. Exercícios