A Linguagem de Programação R: Aplicação em estudos florestais

MSc. *José Wesley Lima Silva*

Conceitos de Programação com R

DIA 01

Recife, 26 de Novembro de 2019

Sumário

Software ou Linguagem de Programação?

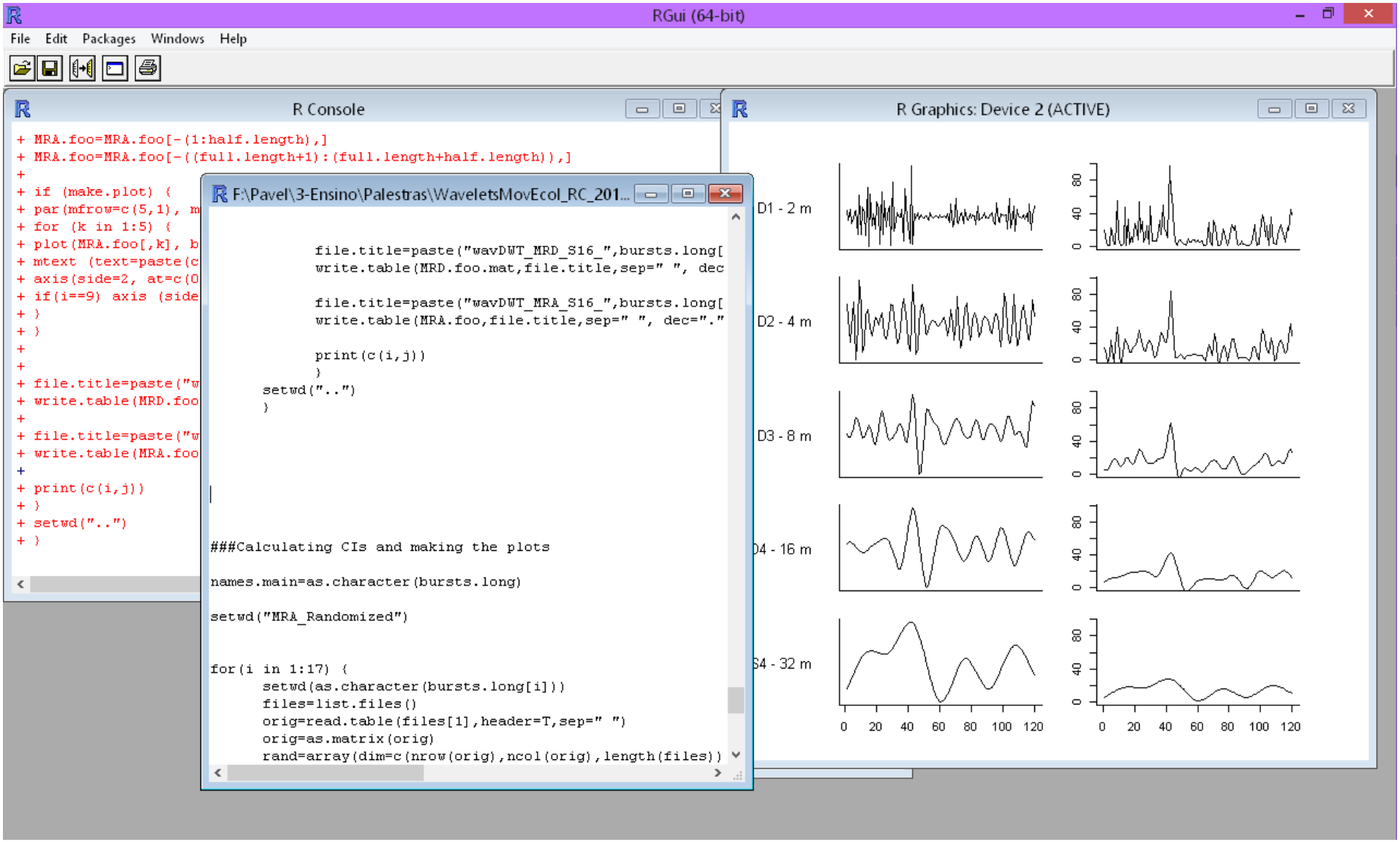
De onde Surgiu?

- Criado em 1993;
- Ross Ihaka e Robert Gentleman (DE. Universidade de Aucklan, Nova Zelândia);
- Linguagem S e Scheme;
- Interpretada, procedural;
- Orientada a objetos;
- *Software Livre* (1995);
- *R Core Team*.

PYPL Index (Worldwide)

May 2019	Change	Programming language	Share	Trends
1		Python	27.34 %	+4.5 %
2		Java	20.25 %	-2.1 %
3		Javascript	8.51 %	-0.0 %
4	↑	C#	7.38 %	-0.5 %
5	↓	PHP	7.34 %	-0.9 %
6		C/C++	6.01 %	-0.3 %
7		R	4.16 %	-0.1 %
8		Objective-C	2.91 %	-0.6 %
9		Swift	2.5 %	-0.3 %
10		Matlab	2.03 %	-0.3 %
11	↑	TypeScript	1.61 %	+0.1 %
12	↓	Ruby	1.4 %	-0.3 %
13		VBA	1.38 %	-0.0 %
14	↑↑↑	Go	1.22 %	+0.4 %
15	↑	Kotlin	1.2 %	+0.3 %
16	↓	Scala	1.17 %	-0.0 %
17	↓↓↓	Visual Basic	1.09 %	-0.2 %
18	↑↑	Rust	0.63 %	+0.3 %
19	↓	Perl	0.61 %	-0.1 %
20	↓	Lua	0.4 %	+0.0 %
21		Haskell	0.32 %	+0.0 %
22	↑	Julia	0.27 %	+0.1 %
23	↓	Delphi	0.26 %	-0.0 %

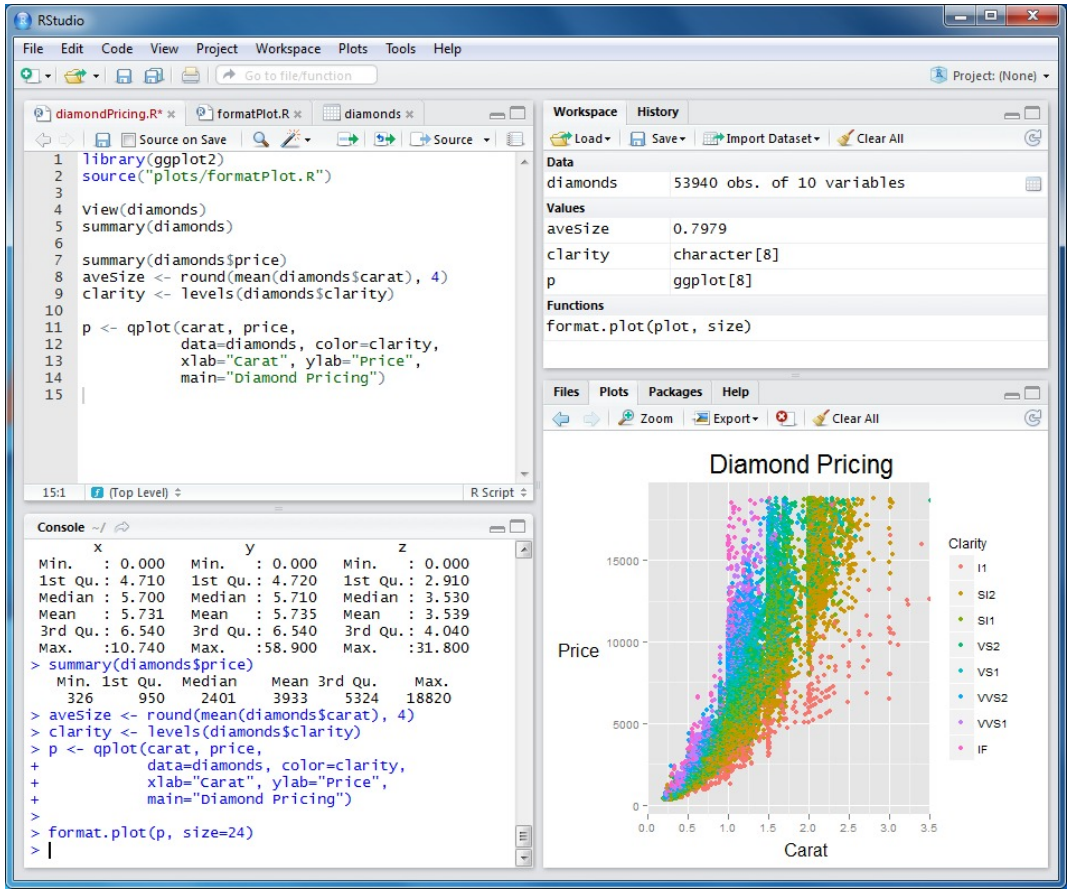
Analista Raiz...



Integrated Development Environment (IDE)

Codar de forma intuitiva

- Criado em 2010;
- Joseph J. Allaire;
- Mais que um bloco de notas...
- Um poderoso editor de texto;
- Organização;
- Praticidade;
- Integração;
- Como criar um projeto no RStudio?



Um amontoado de pacotes

- A linguagem **R** é composta de **três** partes básicas:
  - o **R-base** - o "coração" do R que contém as funções principais, inicia automaticamente;
  - os **pacotes recomendados** - instalados junto com o R-base mas não carregam automaticamente;
  - os **pacotes contribuídos** - não são instalados junto com o R-base;
  - Total de pacotes oficiais - *14.382*.

```
1  #Instalar um pacote
2  install.packages("package_name", dependencies=TRUE)
3
4  #Exemplo
5  pack_nome <- c("dplyr", "tidyr", "ggplot2")
6  install.packages(pack_nome, dependencies = T)
7
8  #Carregar o pacote para ser utilizado
9  library("package_name")
10 require("package_name")
11
12 #Verificar os pacotes carregados na memoria
13 search() #isso e um comentario
```

Help and Error

Help me, please...

```
#tudo sobre o comando no seu navegador
??sequence

#ajuda do comando especifico
?seq
help(seq)

#mostrar exemplos do comando
example(seq)

#comunidade R e Stack Overflow
```

Não sei o que deu errado!

```
#lendo saida de erros
print"Hello world")
Erro: unexpected string constant in "print"Hello world""
#Warning message:
R graphics engine version 12 is not supported by this version of RStudio. The
     Plots tab will be disabled until a newer version of RStudio is installed.
```

Manipulando objetos

Criando objetos

```
1  #o que e um objeto ou variavel?
2  x      <- 9
3  raiz_2 <- sqrt(x)
4
5  #forma errada de criar objeto
6  1   <- 10
7  1_x <- 10
```

Removendo objetos

```
1  A <- 1  #cria o objeto A
2  B <- 2  #cria o objeto B
3
4  #Remover com a funcao rm
5  rm(A,B)
```

Atributos dos objetos

Tipos de dados

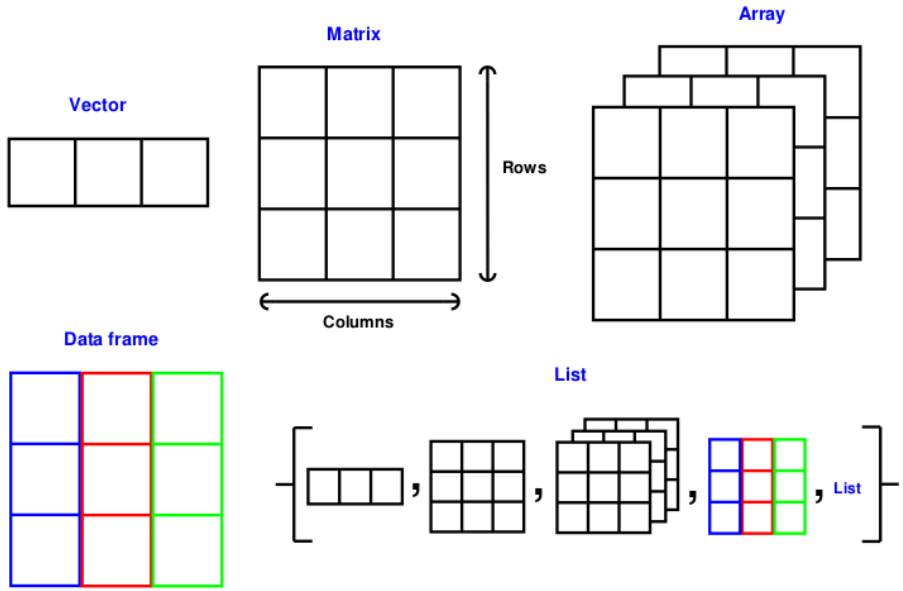
- **character** - textos ou caracteres;
- **numeric** - números inteiros ou reais;
- **logical** - verdadeiro ou falso (TRUE/FALSE);
- **complex** - números complexos;
- **list** - combina diferentes tipos num mesmo objeto;
- **function** - comandos.

Tipos de objetos

```
1   #verificar o tipo de objeto
2
3   is.character()
4   is.numeric()
5   is.logical()
6   is.na()
7   is.vector()
8   is.matrix()
9
10  #transformacoes
11
12  numeric()
13  factor()
14  as.vector()
15  as.factor()
16  as.matrix()
17  as.array()
18  as.list()
```

Tipos de estruturas

- **vector** - vetor com um ou mais elementos, uma dimensão;
- **matrix** - é uma matriz, duas dimensões;
- **array** - pode conter um (vetor), duas (matriz) ou mais dimensões;
- **fator** - vetor que representa dados categóricos;
- **data.frame** - parece com uma matriz, mas permite colunas de diferentes tipos em um mesmo objeto;
- **list** - objeto que permite combinar diferentes estruturas de dados num único objeto.



Estrutura de dados

Vetor

```r
idade <- c(10, 15,18, 12, 90)
nomes <- c('joao', 'maria', 'ana')
vetor.vazio <- c()

#O que acontece?
idade_nomes <- c('joao', 15, 'maria', 16, 'ana', 17)

#Atributos de um objeto

#modo de armazenamento
mode(idade)
mode(nomes)

#Numero de elementos
length(nomes)

#Classe
class(idade)

#indice de um vetor
idade[1]
nomes[2:3]
```

Matriz

```r
m1 <- matrix(data=1:12,nrow=3,ncol=4, byrow=T)
m2 <- matrix(1:12, nrow = 4)

#nomeando
rownames(m1) <- c("R1","R2","R3")
colnames(m1) <- c("C1","C2","C3","C4")

#Algebra Matricial
#multiplicacao comun
n <- c(1,2,3)
m1 * n

#multiplicacao matricial
m1 \% * \% n

#Diagonal da matriz
diag(m1*n)

#indice de matriz
m1[linha,coluna]
m1[1,1] ou m1[1:2,2:3]
```

Estrutura de dados

Array

```
#array

ar1 <- array(1:24, dim = c(3, 4, 2)
     )


#Consultas

ar1[linha,coluna,dimensao]
ar1[, 3:4, ]
ar1[,,2]

#Operacoes matematicas

sum(ar1[, , 1])
mean(ar1[, 2, 1])
```

Lista

```
#formas para criar listas

lista1 <- list('joao', 15, 'maria',
      16, 'ana', 13)
lista2 <- list(c('joao', 'maria', '
     ana'),  c(15, 16, 13))
lista3 <- list(nomes=c('joao', '
     maria', 'ana'),
         idades=c(15, 16, 13))

#indices
lista3[[elemento1]][indice-
     elemento1]
lista3[[1]]
lista3$nomes
lista3[[1]][2]
```

Data Frame

```
#no data frame cada coluna e tratada separadamente
#dentro de uma mesma coluna todos elementos tem que ser do mesmo tipo
d1 <- data.frame(nome=c('joao','maria','jose'),
         sexo=c('mas', 'fem', 'mas'),
         idade=c(20, 21, 22))

#atributos

str(d1)
names(d1)
attach(d1)
head(d1)      #exibe as 5 primeiras linhas
tail(d1)      #exibe as ultimas 5 linhas
d1$cols       #acessa as colunas do banco de dados
d1$col1[1]    #acessa o primeiro elemento da coluna 1

#estatisticas

summary(d1)   #exibe estatisticas descritivas para as colunas do df
```

Infinito e Ausente

Elementos especiais

- **Inf** - 2/0 = Inf - valor que tende ao infinito;
- **NaN** - Not a Namber;
- **NA** - Not Available, um dado ausente;

Operadores Lógicos

```
1       ==      #igual
2       !=      #diferente
3       <,>     #menor que, maior que
4       <=,>=   #menor ou igual a, maior ou igual a
5       |       #logico OU
6       &       #logico E
7       !       #logico de negacao NAO
```

Calculando...

Start R!

```
#estatistica

max(), min(), range()   #maximo, minimo, amplitude
which.max(x)            #indice do maior valor de x
which.min(x)            #indice do menor valor de x
sum(x)                  #soma dos elementos de x
prod(x)                 #produto dos elementos de x
mean(x), var(x)         #media e variancia dos elementos de x
median(x)               #mediana de x
order(x)                #vetor contendo as posicoes ordenadas crescentes de x
rank(x)                 #vetor com ranqueamento de x
sort(x)                 #versao ordenada crescente de x
```

Como importar uma base dados?

Funções para leitura de arquivos

```
#funcoes
read.table()
read.csv()
read.xlsx()   #funcao externa
```

Lendo arquivos

```
#read
dados <- read.csv(                  #le dados de um arquivo csv
    "pasta1/nome_aqruivo.csv",      #endereco do arquivo
    header=T,                       #primeira linha e cabecalho
    sep=","                         #virgula como tabulador - separa colunas
)

dados['nova_coluna'] <- 1 #adiciona uma nova coluna preenchida de 1
write.csv(dados, "novo_arquivo.csv") #escrever um  arquivo
```

Apply

A família Apply representa um conjunto de funções básicas do R que permite realizar operações sobre os dados contidos nas várias estruturas disponíveis (vetor, data frame, listas).

- apply() - aplica uma função a um data frame ou matriz sobre linhas ou colunas;
- lapply(); sapply() - aplica uma função aos elementos de uma lista;
- tapply() - aplica uma função pelo nível de um fator em um data frame;

Apply

```
1   #apply
2   df.a1 <- df[, 5:6]
3   apply(df.a1, 2, mean) # 1 = linha  2 = coluna
4   apply(df.a1, 2, sd)
5   apply(df.a1, 2, var)
6
7   #lapply
8   lapply(list(alt = df$altura, dap = df$dap), mean)
9
10  #tapply
11  tapply(df$altura, df$espacamento, mean)
12  tapply(df$dap, df$clone, mean)
```

O pacote dplyr

O dplyr é o pacote mais indicado para realizar transformação em banco de dados, ele une simplicidade e eficiência de forma elegante.

- filter() - escolhe linhas com base em seus valores (seleciona linhas);
- select() - seleciona colunas;
- mutate() - cria ou modifica colunas;
- arrange() - ordena o banco de dados;
- summarise() - reduz vários valores para um único resumo;

- left_join(a, b, by = "x1"): junta as colunas da tabela "b"na tabela "a". Correspondente ao mesmo nome;
- right_join(a, b, by = "x1"): junta as colunas da tabela "a"na tabela "b";
- inner_join(a, b, by = "x1"): cria uma tabela somente com as colunas da tabela "b"na tabela "a"que possuam correspondência. Não preenche com NA;
- full_join(a, b, by = "x1"): cria uma tabela juntando a tabela "b"na tabela "a"independente de existir correspondência. Preenche com NA.

O pacote dplyr

```
#filter
df.e1 <- filter(df, (espacamento == 'E1')) #linhas E1 na coluna espacamento

df.e1.c1 <- df %>%
        filter(espacamento == 'E1') %>%
        filter(clone == 'C1')

#select
df2 <- df %>%
       select(espacamento, clone, altura, dap)

#mutate
df3 <- df %>%
      mutate(codigo = paste(espacamento, clone, repeticao, sep=""),
      volume = (dap^2*pi/40000)*altura*0.51)

#arrange
df4 <- df %>%
      arrange(arvoresh)  #se decrescente use desc(coluna)

#summarise
df5 <- df3 %>%
      group_by(espacamento, clone, repeticao) %>%
      summarise(narvores=unique(arvoresh), altura_m = mean(altura),
      dap_m = mean(dap), volume_m = mean(volume))
```

O pacote tidyr

- spread() - converte linhas em colunas;
- gather() - converte colunas em linhas.

O pacote tidyr

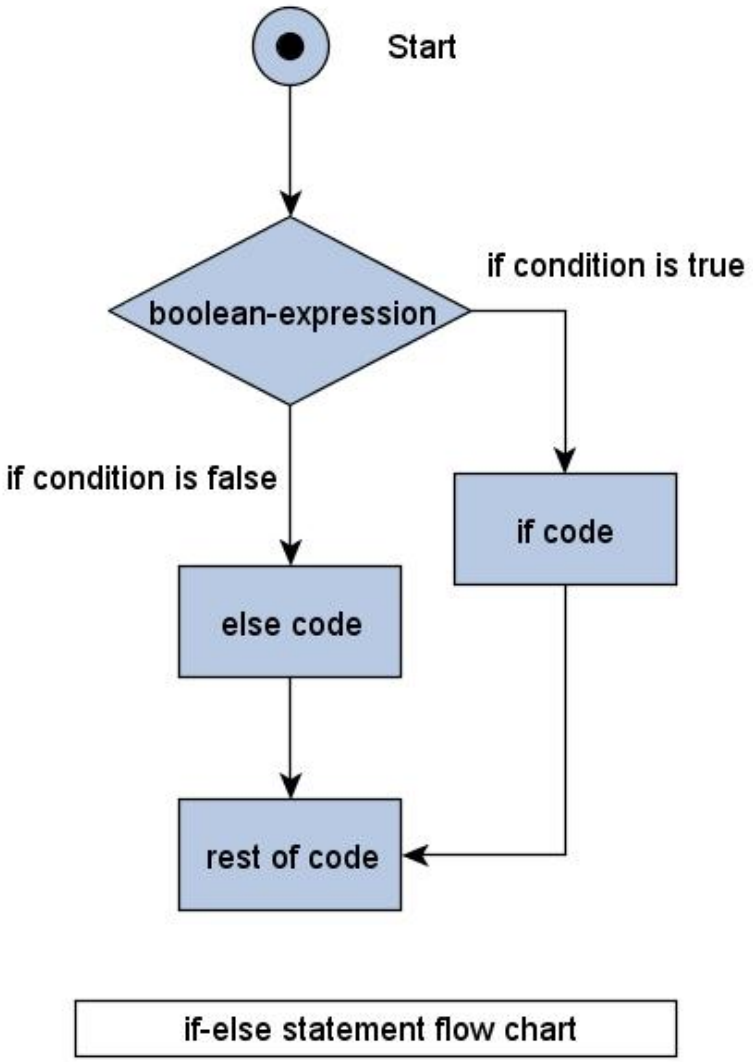
```
#spread
df6 <- df5 % >%
         select(espacamento, clone, repeticao, dap_m) % >%
         spread(repeticao, dap_m)

#gather
df6 % >%
     gather(repe, dap, -1:-2)
```

Estruturas condicionais

if, else - se, senão

```
1   if     #avalia uma condicao e executa uma
           expressao, se condicao verdadeira
2   else   #caso a condicao inicial seja
           falsa execute este comando
3   ifelse #uma forma vetorizada do if else
4
5   #exemplos
6   #comandos rapidos
7   a <-1; b <- 5
8   if(a<b) print("hello world")
9   if(a>b) print("hello world") else print("
        bye")
10
11  #forma usual
12  if(a>0 & a <= 1){
13    pint("hello world")
14  }else{
15    print("bye")
16  }
```



if-else statement flow chart

Estruturas de repetição

loop - laço

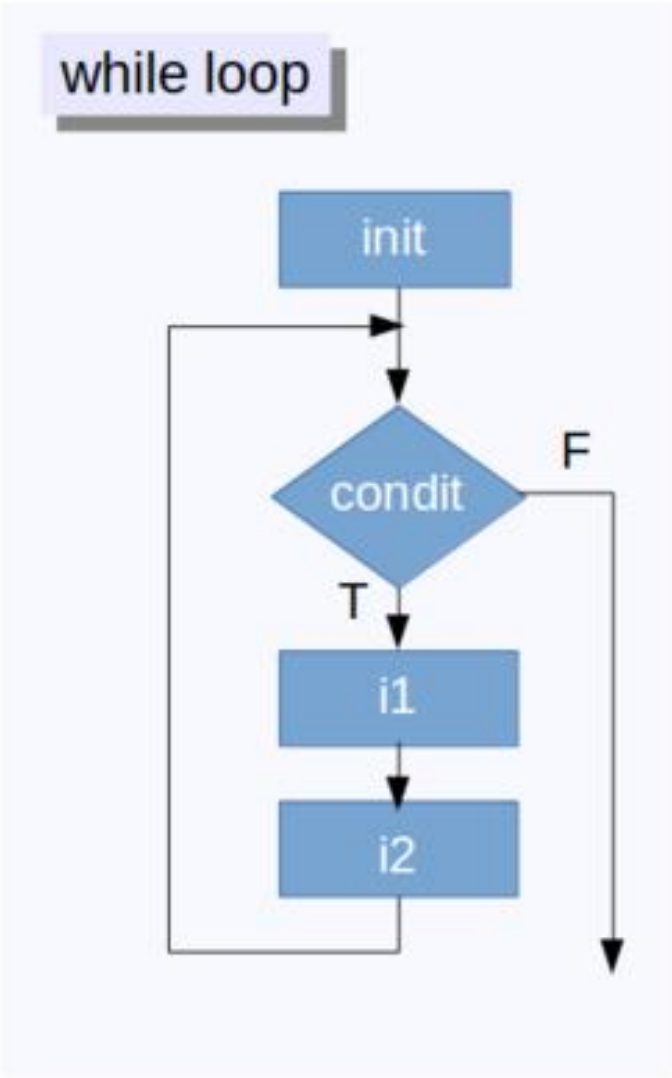
```
repeat   #executa uma repeticao indefinidamene, precisa do comando break
for    #pecorre um conjunto de valores, executando uma tarefa
while  #repete enquanto a condicao expressa no argumento for verdadeira
break  #interrompe o loop
next   #execute uma proxima vez
```

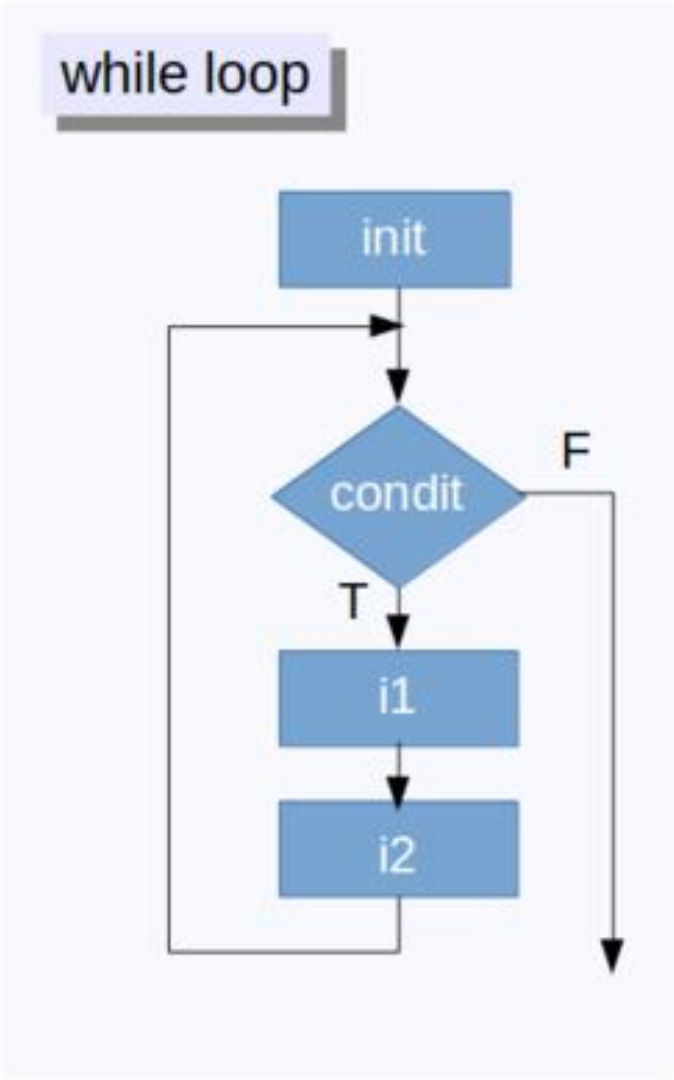
repeat

```
#cuidado com o infinito
a <- 0

repeat{
  if(a==5){
    break
  }else{
    a <- a + 1
  }
}
```

Estruturas de repetição

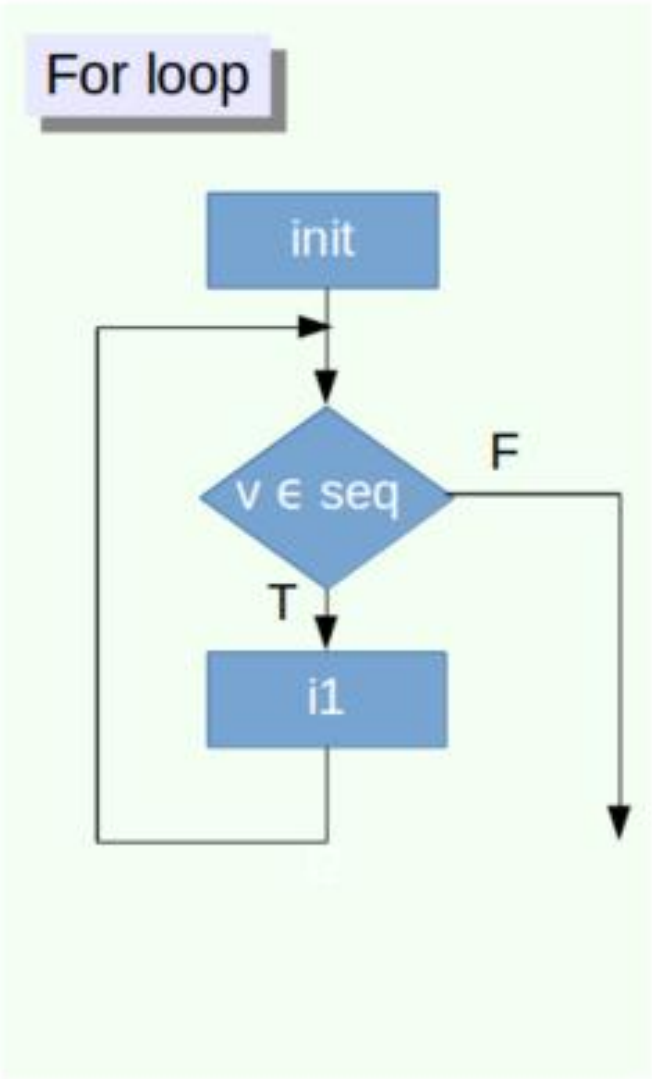




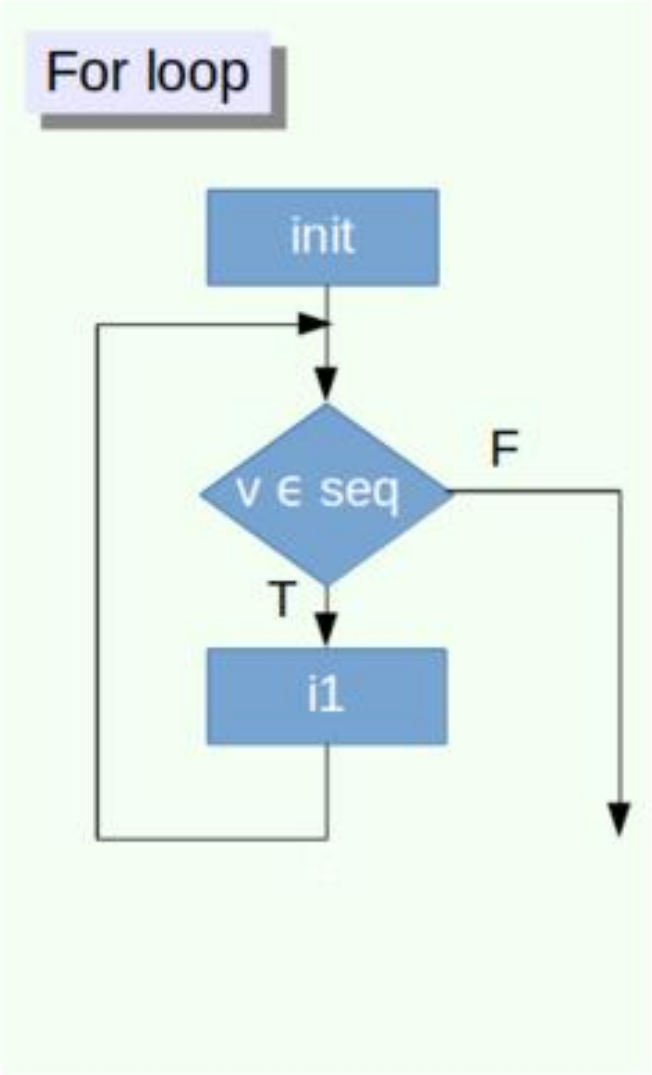
while - enquanto

```
1   #enquanto verdade execute
2
3   a <- 0
4
5   while(a < 10){
6   print(a)
7   a <- a + 1
8   }
9
10  #usando while de forma interativa
11  x <- 1
12
13  while(x != 0){
14    x <- scan()
15    print(paste("Voce digitou: ", x))
16    print("Para sair digite: 0")
17  }
```

Estruturas de repetição



Estruturas de repetição



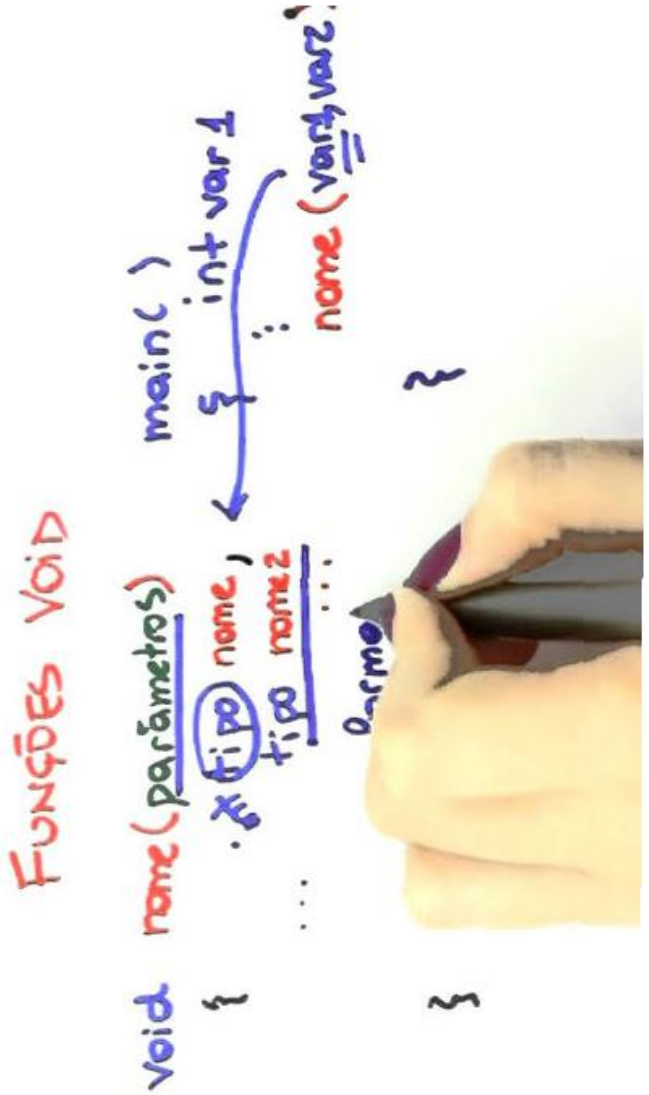
for - para

```
1   #para isso, faca aquilo
2
3   a <- [1:10]
4
5   for(i in a){
6   print(i)
7   }
8
9   #soletrando
10  nomes <- c("joao", "maria", "ana", "
        jose")
11  for(nome in nomes){
12    print(nome)
13    for(letra in strsplit(nome, "")){
14      print(letra)
15    }
16  }
```

Não refaça a roda...

função

```
1   function   #sao blocos de codicos
         cujo objetivo principal e a
         reutilizacao
2   return     #retorna a saida de uma
         funcao
3
4   #em uma linha
5   par <- function(x) x%%2 == 0
6   par(2)
7
8   #usualmente
9   par <- function(x){
10  if (!is.integer(x)){
11  warning("Valor nao inteiro")
12  }else{
13  p <- x%%2 == 0
14  return(p)
15  }
16  }
```



A Linguagem de Programação R: Aplicação em estudos florestais

MSc. *José Wesley Lima Silva*

Conceitos de Programação com R

DIA 01

Recife, 26 de Novembro de 2019