Estruturas de Controle

Iven Valpassos

29/07/2021

Todos os dias tomamos decisões do tipo “SE”/“ENTÃO” .

Se o sinal está vermelho, pare o carro, se está verde siga em frente.

O despertador tocou, levante da cama se tiver reunião cedo, se não tiver reunião durma mais 15 min… e por aí vai.

Utilizar estruturas de controle em programação é traduzir estas decisões em um formato que o computador entenda. Não tem mistério!

x = 30  
if (x < 45)   
 {"Este número é menor que 45"}

## [1] "Este número é menor que 45"

Criamos uma variável (x) igual a 30 e pedimos para o R imprimir “Este número é menor que 45” se x for menor do que 45.

Nós colocamos a ação que o R precisa executar entre chaves. Isso não é obrigatório, mas é uma boa prática. Veja que o código funciona mesmo sem as chaves.

if (x < 45)   
 "Este número é menor que 45"

## [1] "Este número é menor que 45"

Agora, imagine que que você quer executar uma ação caso a condição seja verdadeira, no entanto, se a condição não for satisfeita o R deve executar outra ação. Neste caso você vai usar uma estrutura ‘if’/‘else’ (‘se’/‘caso contrário’).

if (x < 7) {  
 "Este número é menor que 7"  
} else {  
 "Este número não é menor que 7"  
 }

## [1] "Este número não é menor que 7"

As estruturas de controlhe podem ser aninhadas em casos com maior complexidade

x<- 86  
if (x<50){  
 'Este número é menor que cinquenta'  
}else if (x==50)   
 {'Este número é igual a 50'  
} else {'Este número é maior que 50'}

## [1] "Este número é maior que 50"

Podemos utilizar uma função que torna o código mais simples.

x<- 120  
  
ifelse(x<100, 'O valor é menor que 100','Não sei informar')

## [1] "Não sei informar"

x<- c(1,3,15,25,44,50)  
ifelse(x%%2==0, 'O número é par.',  
 ifelse(x%%3==0 , 'O número é múltiplo de 3',NA))

## [1] NA "O número é múltiplo de 3"  
## [3] "O número é múltiplo de 3" NA   
## [5] "O número é par." "O número é par."

Vimos no exemplo acima que a função ‘ifelse’ faz uma iteração através dos elementos de x e retorna uma frase caso o número seja par e outra frase quando o número é múltiplo de 3.

Podemos usar o loop ‘for’ para fazer este tipo de operação.

for (i in 1:20) {print(i)}

## [1] 1  
## [1] 2  
## [1] 3  
## [1] 4  
## [1] 5  
## [1] 6  
## [1] 7  
## [1] 8  
## [1] 9  
## [1] 10  
## [1] 11  
## [1] 12  
## [1] 13  
## [1] 14  
## [1] 15  
## [1] 16  
## [1] 17  
## [1] 18  
## [1] 19  
## [1] 20

for (i in rep(rnorm(10))){print(i\*2)}

## [1] -1.586537  
## [1] 1.047268  
## [1] 1.127805  
## [1] 2.422907  
## [1] 0.435266  
## [1] 1.329232  
## [1] -1.313527  
## [1] -1.071113  
## [1] 3.54927  
## [1] -2.497038

Podemos fazer com que o loop for ignore alguns elementos durante a iteração.

for (i in 1:100){  
 if (i%%2==0|i%%3==0)  
 next  
 print(i)  
}

## [1] 1  
## [1] 5  
## [1] 7  
## [1] 11  
## [1] 13  
## [1] 17  
## [1] 19  
## [1] 23  
## [1] 25  
## [1] 29  
## [1] 31  
## [1] 35  
## [1] 37  
## [1] 41  
## [1] 43  
## [1] 47  
## [1] 49  
## [1] 53  
## [1] 55  
## [1] 59  
## [1] 61  
## [1] 65  
## [1] 67  
## [1] 71  
## [1] 73  
## [1] 77  
## [1] 79  
## [1] 83  
## [1] 85  
## [1] 89  
## [1] 91  
## [1] 95  
## [1] 97

Podemos interromper a execução do loop for se alguma condição for satisfeita.

x<-1  
  
for (i in 1:20){  
 x = x + i  
 if(x>50)  
 break  
 print(x)  
}

## [1] 2  
## [1] 4  
## [1] 7  
## [1] 11  
## [1] 16  
## [1] 22  
## [1] 29  
## [1] 37  
## [1] 46

Em algumas situações, queremos que o R repita uma determinada operação até que uma condição de parada seja satisfeita. Uma das formas de fazer isso é com a função ‘repeat’.

x<- 1  
y<-1  
repeat{  
 x = x\*1.01  
 y = y +1  
if (x>1.30)  
 break  
print(x)  
print(y)}

## [1] 1.01  
## [1] 2  
## [1] 1.0201  
## [1] 3  
## [1] 1.030301  
## [1] 4  
## [1] 1.040604  
## [1] 5  
## [1] 1.05101  
## [1] 6  
## [1] 1.06152  
## [1] 7  
## [1] 1.072135  
## [1] 8  
## [1] 1.082857  
## [1] 9  
## [1] 1.093685  
## [1] 10  
## [1] 1.104622  
## [1] 11  
## [1] 1.115668  
## [1] 12  
## [1] 1.126825  
## [1] 13  
## [1] 1.138093  
## [1] 14  
## [1] 1.149474  
## [1] 15  
## [1] 1.160969  
## [1] 16  
## [1] 1.172579  
## [1] 17  
## [1] 1.184304  
## [1] 18  
## [1] 1.196147  
## [1] 19  
## [1] 1.208109  
## [1] 20  
## [1] 1.22019  
## [1] 21  
## [1] 1.232392  
## [1] 22  
## [1] 1.244716  
## [1] 23  
## [1] 1.257163  
## [1] 24  
## [1] 1.269735  
## [1] 25  
## [1] 1.282432  
## [1] 26  
## [1] 1.295256  
## [1] 27

O loop ‘while’ também pode ser utilizado quando você deseja que que uma operação seja executada até que uma condição seja satisfeita.

x<- 1  
y<- 1  
while (x<1.30){  
 x = x\*1.01  
 y = y+1  
 print(x)  
 print(y)  
}

## [1] 1.01  
## [1] 2  
## [1] 1.0201  
## [1] 3  
## [1] 1.030301  
## [1] 4  
## [1] 1.040604  
## [1] 5  
## [1] 1.05101  
## [1] 6  
## [1] 1.06152  
## [1] 7  
## [1] 1.072135  
## [1] 8  
## [1] 1.082857  
## [1] 9  
## [1] 1.093685  
## [1] 10  
## [1] 1.104622  
## [1] 11  
## [1] 1.115668  
## [1] 12  
## [1] 1.126825  
## [1] 13  
## [1] 1.138093  
## [1] 14  
## [1] 1.149474  
## [1] 15  
## [1] 1.160969  
## [1] 16  
## [1] 1.172579  
## [1] 17  
## [1] 1.184304  
## [1] 18  
## [1] 1.196147  
## [1] 19  
## [1] 1.208109  
## [1] 20  
## [1] 1.22019  
## [1] 21  
## [1] 1.232392  
## [1] 22  
## [1] 1.244716  
## [1] 23  
## [1] 1.257163  
## [1] 24  
## [1] 1.269735  
## [1] 25  
## [1] 1.282432  
## [1] 26  
## [1] 1.295256  
## [1] 27  
## [1] 1.308209  
## [1] 28

Agora vamos ver um exemplo utilizando um data frame.

petr4<- read.csv('petr4.csv')  
head(petr4)

## X company\_name date time\_begin time\_end open high low close  
## 1 2778 PETROBRAS 2015-06-30 10:00:00 10:09:59 12.90 12.96 12.82 12.95  
## 2 2779 PETROBRAS 2015-06-30 10:10:00 10:19:59 12.94 12.94 12.85 12.88  
## 3 2780 PETROBRAS 2015-06-30 10:20:00 10:29:59 12.89 12.89 12.77 12.86  
## 4 2781 PETROBRAS 2015-06-30 10:30:00 10:39:59 12.86 12.99 12.82 12.87  
## 5 2782 PETROBRAS 2015-06-30 10:40:00 10:49:59 12.86 12.87 12.76 12.83  
## 6 2783 PETROBRAS 2015-06-30 10:50:00 10:59:59 12.83 12.88 12.78 12.85  
## mid\_price quantity vwap twap volume oscilacao notional  
## 1 12.88875 1329 12.90 12.91 2708800 0.003875969 34930623  
## 2 12.88835 1496 12.89 12.90 2080600 -0.004636785 26814724  
## 3 12.81933 1436 12.82 12.85 1667900 -0.002327386 21380527  
## 4 12.91699 3139 12.91 12.88 3809800 0.000777605 49181078  
## 5 12.81388 2386 12.81 12.83 2502800 -0.002332815 32059292  
## 6 12.83510 1304 12.84 12.84 1077200 0.001558846 13827478

library(zoo)

##   
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':  
##   
## as.Date, as.Date.numeric

library(dplyr)

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

library(tidyr)  
library(data.table)

##   
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':  
##   
## between, first, last

Nosso primeiro passo é criarmos uma série com a média móvel com janela mais curta (sma) e uma séria com a média móvel mais longa (lma) a partir do preço de fechamento de dada linha da nossa série.

petr4<- petr4%>%mutate(sma = rollapply(petr4['close'],FUN = mean,width = 9,fill = NA,align = 'right'))  
petr4<- petr4%>%mutate(lma = rollapply(petr4['close'],FUN = mean,width = 18,fill = NA,align = 'right'))

Agora vamos criar mais duas séries. Em cada uma delas vamos ter o valor calculado para a média móvel no instante anterior.

petr4<- petr4%>%mutate(previous\_sma= shift(petr4$sma,n = 1, type = 'lag'))  
petr4<- petr4%>%mutate(previous\_lma = shift(petr4$lma, n = 1, type = 'lag'))

Vamos utilizar as séries criadas anteriormente para gerarmos sinais de compra e venda da ação da Petrobras (vamos ver maiores detalhes no nosso estudo de caso, não se preocupe caso ainda não esteja 100% claro pra você o que estamos fazendo).

petr4<- drop\_na(petr4)  
petr4$signal<- NA

inicio <- proc.time()  
for (i in 1:nrow(petr4)){  
 if (petr4[i,'sma']>petr4[i,'lma']&petr4[i,'previous\_sma']<petr4[i,'previous\_lma']){  
 petr4[i,'signal']<- 'Buy'  
   
 }else if(petr4[i,'sma']<petr4[i,'lma']&petr4[i,'previous\_sma']>petr4[i,'previous\_lma']){  
 petr4[i,'signal']<- 'Sell'  
 }  
}  
proc.time()-inicio

## user system elapsed   
## 5.93 0.55 6.47

Vamos carregar os dados novamente e calcular o sinal utilizando uma função da família ‘apply’.

<https://medium.com/@fernando.gama/fam%C3%ADlia-apply-fugindo-um-pouco-dos-loops-convencionais-43ff6ba96013>

petr4<- read.csv('petr4.csv')  
  
petr4<- petr4%>%mutate(sma = rollapply(petr4['close'],FUN = mean,width = 9,fill = NA,align = 'right'))  
petr4<- petr4%>%mutate(lma = rollapply(petr4['close'],FUN = mean,width = 18,fill = NA,align = 'right'))  
  
petr4<- petr4%>%mutate(previous\_sma= shift(petr4$sma,n = 1, type = 'lag'))  
petr4<- petr4%>%mutate(previous\_lma = shift(petr4$lma, n = 1, type = 'lag'))  
  
petr4<- drop\_na(petr4)  
petr4$signal<- NA

Criando a função que calcula o sinal.

calcula\_sinal <- function(sma,lma,previous\_sma,previous\_lma){  
 if(sma>lma & previous\_sma < previous\_lma){  
 return('Buy')  
 }else if(sma<lma & previous\_sma > previous\_lma)  
 return('Sell')  
   
}

Usando a função ‘mapply’.

inicio <- proc.time()  
  
petr4$signal<- mapply(FUN=calcula\_sinal,sma = petr4$sma,lma = petr4$lma,previous\_sma = petr4$previous\_sma,previous\_lma = petr4$previous\_lma)  
  
proc.time()-inicio

## user system elapsed   
## 0.17 0.00 0.17