Módulo 6 - Funções + Data e Hora

Marcus Suassuna Santos

26/10/2020

Funções

- Vimos uma série de funções, mas agora vamos apreseder a escrever nossas próprias funções para atender a nossas finalidades
- Atividade central da programação (em R, mas penso que em qualquer linguagem)
- Progresso de USUÁRIO -> PROGRAMADOR
- Funções: executam rotinas que são aplicáveis a outros dados ou contextos semelhantes
- Funções ligam dados e parâmetros de entrada a uma saída específica, liberando o usuário dos detalhes de programação

Funções

Pra iniciar, vamos criar nossa primeira função:

```
funcao <- function(){
  # Função vazia
}
class(funcao)
```

```
## [1] "function"
```

```
funcao()
```

```
## NULL
```

Funções

Pra iniciar, vamos criar uma primeira função:

```
funcao1 <- function(){
  # Função vazia
}
class(funcao)
```

```
## [1] "function"
```

```
funcao1()
```

```
## NULL
```

Não é muito interessante, contém apenas o NOME da função

Funções

Inserindo um CORPO da função:

```
funcao2 <- function(){

  print("Minha segunda função")

}
funcao2()
```

```
## [1] "Minha segunda função"
```

Contém NOME e CORPO

Funções

Inserindo ARGUMENTOS na função:

```
funcao3 <- function(num){

  for(i in seq_len(num)){
    print("Minha terceira função!")
  }

}
funcao3(3)
```

```
## [1] "Minha terceira função!"
## [1] "Minha terceira função!"
## [1] "Minha terceira função!"
```

Contém NOME, CORPO e ARGUMENTOS

Funções

Inserindo um RETORNO na função:

```
funcao4 <- function(num){
  for(i in seq_len(num)) print("Minha quarta função!")
  numQuadrado <- num ^ 2
}
resultado <- funcao4(2)
```

```
## [1] "Minha quarta função!"
## [1] "Minha quarta função!"
```

```
resultado
```

```
## [1] 4
```

Contém NOME, CORPO, ARGUMENTOS e RETORNO (implícito)

Funções

Exemplo: estatística de teste t-Student

```
tStudent_Stat <- function(r, n){
  return(r * sqrt(n-2) / sqrt(1-r^2))
}
```

```
tStudent_Stat(0.8, 40)
```

```
## [1] 8.219219
```

```
qt(0.025, 40-2)
```

```
## [1] -2.024394
```

```
abs(tStudent_Stat(0.8, 40)) > qt(0.025, 40-2)
```

```
## [1] TRUE
```

Funções

Na função acima, caso nãos e especifique o argumento `num`, ela retorna um erro. É possível especificar argumentos *dafault*, caso exista interesse.

```
funcao5 <- function(num = 1){
  for(i in seq_len(num)) print("Minha quinta função!")
  numQuadrado <- num ^ 2
}
resultado <- funcao5()
```

```
## [1] "Minha quinta função!"
```

```
resultado
```

```
## [1] 1
```

Funções

Na função acima, caso nãos e especifique o argumento `num`, ela retorna um erro. É possível especificar argumentos *dafault*, caso exista interesse.

```
funcao5(3)
```

```
## [1] "Minha quinta função!"
## [1] "Minha quinta função!"
## [1] "Minha quinta função!"
```

Funções

Uma função pode ter inúmeros argumentos

```
potencia <- function(a,b){
  a ^ b
}
potencia
```

```
## function(a,b){
##   a ^ b
## }
```

Funções

```
potencia(2,3)
```

```
## [1] 8
```

```
potencia(3,2)
```

```
## [1] 9
```

```
potencia(b = 3, a = 2)
```

```
## [1] 8
```

```
potencia(b = 3, 2)
```

```
## [1] 8
```

Funções

Uma função pode ter inúmeros argumentos

```
potencia2 <- function(a, b, c){
  c * (a ^ b)
}
potencia2
```

```
## function(a, b, c){
##   c * (a ^ b)
## }
```

Funções

Uma função pode ter inúmeros argumentos

```
potencia2(c = 2, 3, 4)
```

```
## [1] 162
```

```
2 * (3 ^ 4)
```

```
## [1] 162
```

Funções

Uma função pode ter inúmeros argumentos

```
args(potencia2)
```

```
## function (a, b, c)
## NULL
```

Funções

Uma função pode ter inúmeros argumentos

```
args(lm)
```

```
## function (formula, data, subset, weights, na.action, method = "qr",
##     model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE,
##     contrasts = NULL, offset, ...)
## NULL
```

Funções

Uma função pode ter inúmeros argumentos

```
args(plot)
```

```
## function (x, y, ...)
## NULL
```

Avaliação *Lazy* no R

No R, um argumento só é avaliado quando necessário. Por exemplo:

```
potencia <- function(a, b, c){
  a ^ b
}
potencia(2,3)
```

```
## [1] 8
```

Avaliação *Lazy* no R

No R, um argumento só é avaliado quando necessário. Por exemplo, aqui ele retornaria um erro ("argumento ausente sem padrão"):

```
#potencia2 <- function(a, b, c){
#  c * (a ^ b)
#}
#potencia2(2, 3)
```

Argumento '…'

O argumento '.' é um argumento especial que indica que existe uma série de outros argumentos que são atribuídos a funções que estão dentro da função criada. Por exemplo, a função `plot`.

```
args(plot)
```

```
## function (x, y, ...)
## NULL
```

Ela tem inúmeros argumentos sob o argumento `...`

Argumento '…'

Por exemplo:

```
plotLinha <- function(x, y, type = "l", ...){
  plot(x, y, type = type, ...)
}
```

Argumento '...'

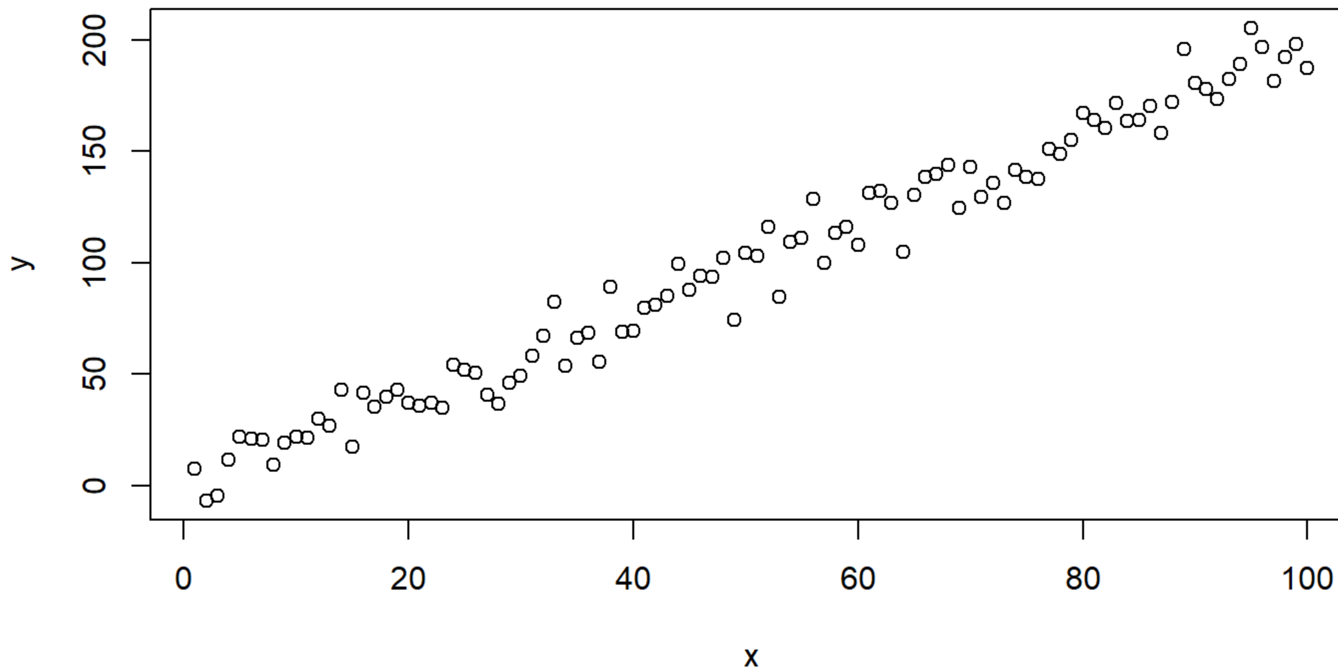
Por exemplo:

```
x <- c(1:100)
y <- 2*x + rnorm(100, 0, 10)
```

Argumento '…'

Por exemplo:

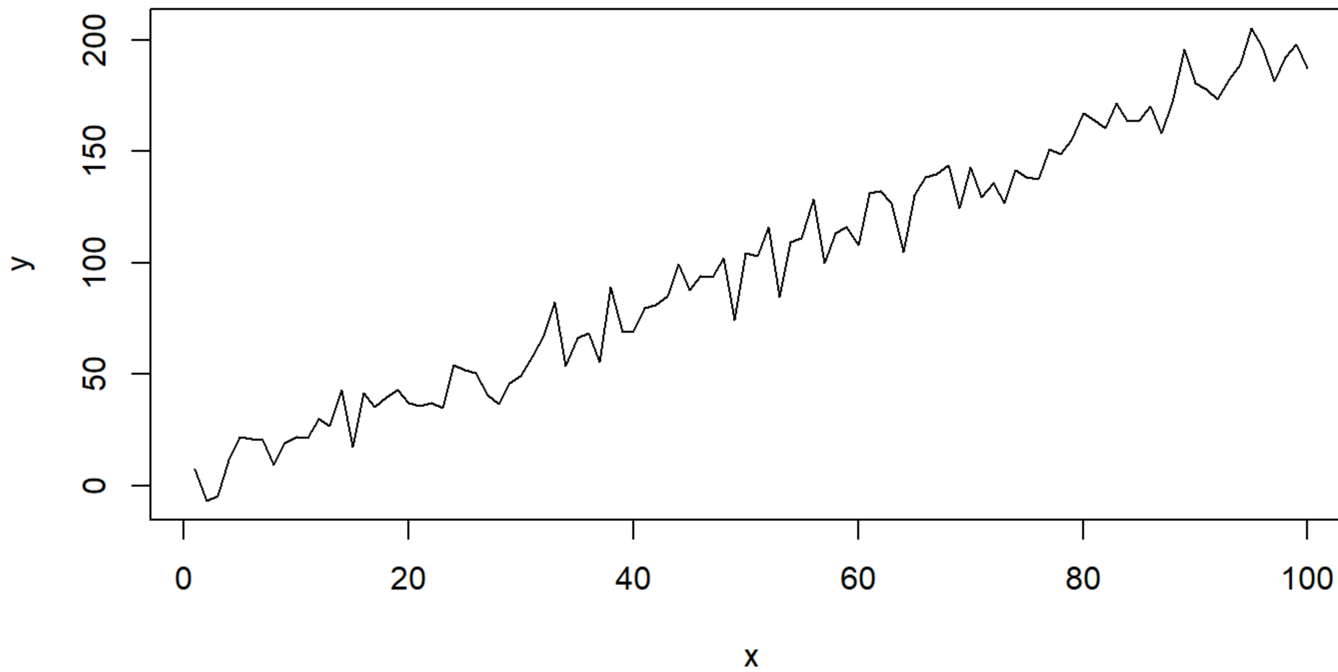
```
plot(x, y)
```



Argumento '…'

Por exemplo:

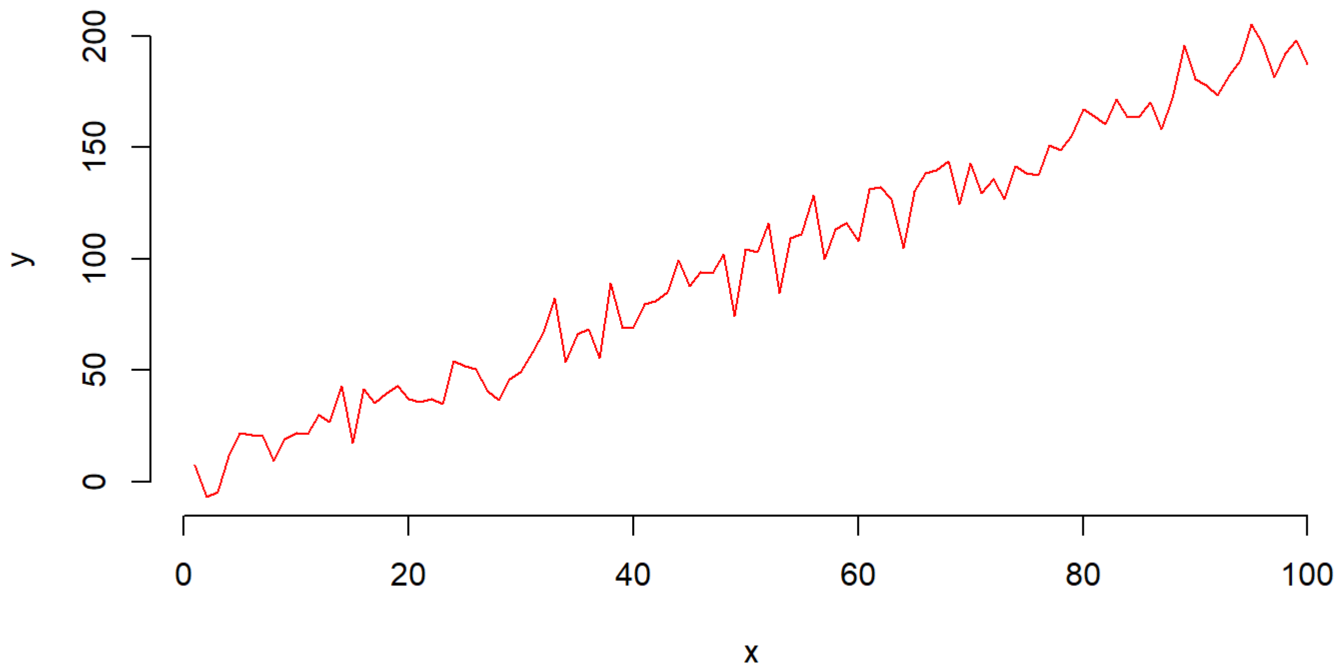
```
plotLinha(x, y)
```



Argumento '...'

Por exemplo:

```
plotLinha(x, y, bty = "n", col = "red")
```



Regras de escopo no R: Escopo léxico (*Lexical Scoping*)

Indica como que a linguagem associa valores às variáveis livres. Pelo escopo léxico, o R faz uma lista de busca de valores associados a um símbolo.

Por exemplo, a função `potencia2`, redefinida abaixo:

Se a função for chamada da forma indicada (`potencia2(2, 3)`), ela retorna um erro porque não encontra um valor nem no ambiente da função nem no ambiente global.

```
potencia2 <- function(a, b){
  c * (a ^ b)
}
# potencia2(2, 3)
```

Regras de escopo no R: Escopo léxico (*Lexical Scoping*)

```
c <- 3
potencia2(2, 3)
```

```
## [1] 24
```

Regras de escopo no R: Escopo léxico (*Lexical Scoping*)

```
c <- 3
potencia2 <- function(a, b, c)  c * (a ^ b)
potencia2(2, 3, 1)
```

```
## [1] 8
```

- EM RESUMO:
- O `R` busca o valor de uma variável no ambiente em que ela foi definida
- Caso ela não encontre, ela busca no ambiente superior (*Global Environment*)
- Caso ela não encontre o valor em nenhum ambiente, a função vai retornar um erro

Datas e Horas no R

O `R` apresente uma representação especial para datas e horas. EM RESUMO:

- Datas são representadas por objetos `Date`
- Horas são representadas por objetos `POSIXct` e `POSIXlt`
- Internamente, objetos do tipo `Date` representam dias a partir de 01/01/1970
- Objetos `POSIXct` e `POSIXlt` representam os segundos a partir de 01/01/1970

Datas e Horas no R

Datas são "convertidas" de variável do tipo `character` para `Date` usando a função `as.Date`

```
x <- "2020-10-26"
print(x)
```

```
## [1] "2020-10-26"
```

```
class(x)
```

```
## [1] "character"
```

Datas e Horas no R

Datas são "convertidas" de variável do tipo `character` para `Date` usando a função `as.Date`

```
x <- as.Date(x)
print(x)
```

```
## [1] "2020-10-26"
```

```
class(x)
```

```
## [1] "Date"
```

```
unclass(x)
```

```
## [1] 18561
```

Datas e Horas no R

Datas são "convertidas" de variável do tipo `character` para `Date` usando a função `as.Date`

```
x <- as.Date(1, origin = "1970-01-01")
print(x)
```

```
## [1] "1970-01-02"
```

```
class(x)
```

```
## [1] "Date"
```

Datas e Horas no R

Funções para extrair informações de datas:

```
weekdays(x)
```

```
## [1] "sexta-feira"
```

```
months(x)
```

```
## [1] "janeiro"
```

```
quarters(x)
```

```
## [1] "Q1"
```

Datas e Horas no R

Funções para extrair informações de datas:

```
weekdays(x)
```

```
## [1] "sexta-feira"
```

```
months(x)
```

```
## [1] "janeiro"
```

```
quarters(x)
```

```
## [1] "Q1"
```

Datas e Horas no R

O formato nativo de horas no R é o `POSIXct`

```
x <- Sys.time()
print(x)
```

```
## [1] "2020-10-19 19:01:55 -03"
```

```
class(x)
```

```
## [1] "POSIXct" "POSIXt"
```

Datas e Horas no R

O formato nativo de horas no R é o `POSIXct`

```
unclass(x)
```

```
## [1] 1603144916
```

Datas e Horas no R

O formato nativo de horas no R é o `POSIXct`

```
x <- as.POSIXlt(x)
names(unclass(x))
```

```
##  [1] "sec"    "min"    "hour"   "mday"   "mon"    "year"   "wday"   "yday"
##  [9] "isdst"  "zone"   "gmtoff"
```

Datas e Horas no R

Coerção de caracteres em datas e horas em objetos do R:

```
x <- as.Date("06/11/2020")
print(x)
```

```
## [1] "0006-11-20"
```

A função as.Date tenta o formato `%Y-%m-%d` e `%Y/%m/%d`. Se o formato for diferente, deve ser explicitado.

Datas e Horas no R

```
as.Date("06/11/2020", "%d/%m/%Y")
```

```
## [1] "2020-11-06"
```

```
as.Date("20201106", "%Y%m%d")
```

```
## [1] "2020-11-06"
```

```
as.Date("201106", "%y%m%d")
```

```
## [1] "2020-11-06"
```

```
as.Date("06nov20", "%d%b%y")
```

```
## [1] "2020-11-06"
```

Datas e Horas no R

Operações com datas e horas: todas as operações matemáticas

```
as.Date("20201106", "%Y%m%d") - as.Date("191106", "%y%m%d")
```

```
## Time difference of 366 days
```

```
as.Date("20201106", "%Y%m%d") > as.Date("191106", "%y%m%d")
```

```
## [1] TRUE
```

```
as.Date("20201106", "%Y%m%d") == as.Date("06/11/2020", "%d/%m/%Y")
```

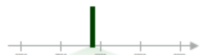
```
## [1] TRUE
```

Pacote **lubridate**

- HELP -> Cheatsheets

Dates and times with lubridate : : CHEAT SHEET

Date-times



2017-11-28 12:00:00

`2017-11-28 12:00:00`
A **date-time** is a point on the timeline, stored as the number of seconds since 1970-01-01 00:00:00 UTC
dt <- as_datetime(1511870400)
"2017-11-28 12:00:00 UTC"

`2017-11-28`
A **date** is a day stored as the number of days since 1970-01-01
d <- as_date(17498)
"2017-11-28"

`12:00:00`
An hms is a **time** stored as the number of seconds since 00:00:00
t <- hms::as.hms(85)
00:01:25

PARSE DATE-TIMES (Convert strings or numbers to date-times)

1. Identify the order of the year (**y**), month (**m**), day (**d**), hour (**h**), minute (**m**) and second (**s**) elements in your data.
2. Use the function below whose name replicates the order. Each accepts a wide variety of input formats.

Example	Function
2017-11-28T14:02:00	**ymd_hms**(), **ymd_hm**(), **ymd_h**(). *ymd_hms("2017-11-28T14:02:00")*
2017-22-12 10:00:00	**ydm_hms**(), **ydm_hm**(), **ydm_h**(). *ydm_hms("2017-22-12 10:00:00")*
11/28/2017 1:02:03	**mdy_hms**(), **mdy_hm**(), **mdy_h**(). *mdy_hms("11/28/2017 1:02:03")*
1 Jan 2017 23:59:59	**dmy_hms**(), **dmy_hm**(), **dmy_h**(). *dmy_hms("1 Jan 2017 23:59:59")*
20170131	**ymd**(), **ydm**(). *ymd(20170131)*
July 4th, 2000	**mdy**(), **myd**(). *mdy("July 4th, 2000")*
4th of July '99	**dmy**(), **dym**(). *dmy("4th of July '99")*
2001: Q3	**yq**() Q for quarter. *yq("2001: Q3")*
2:01	hms::**hms**() Also lubridate::**hms**(), **hm**() and **ms**(), which return periods.* *hms::hms(sec = 0, min= 1, hours = 2)*

Example	Function
2017.5	**date_decimal**(decimal, tz = "UTC") *date_decimal(2017.5)*
	now(tzone = "") Current time in tz (defaults to system tz). *now()*
	today(tzone = "") Current date in a tz (defaults to system tz). *today()*
	fast_strptime() Faster strptime. *fast_strptime('9/1/01', '%y/%m/%d')*
	parse_date_time() Easier strptime. *parse_date_time("9/1/01", "ymd")*

GET AND SET COMPONENTS

Use an accessor function to get a component.
Assign into an accessor function to change a component in place.

d ## "2017-11-28"
day(d) ## 28
day(d) <- 1
d ## "2017-11-01"

Example	Function
2018-01-31 11:59:59	**date**(x) Date component. *date(dt)*
2018-01-31 11:59:59	**year**(x) Year. *year(dt)* **isoyear**(x) The ISO 8601 year. **epiyear**(x) Epidemiological year.
2018-01-31 11:59:59	**month**(x, label, abbr) Month. *month(dt)*
2018-01-31 11:59:59	**day**(x) Day of month. *day(dt)* **wday**(x,label,abbr) Day of week. **qday**(x) Day of quarter.
2018-01-31 11:59:59	**hour**(x) Hour. *hour(dt)*
2018-01-31 11:59:59	**minute**(x) Minutes. *minute(dt)*
2018-01-31 11:59:59	**second**(x) Seconds. *second(dt)*
	week(x) Week of the year. *week(dt)* **isoweek**() ISO 8601 week. **epiweek**() Epidemiological week.
	quarter(x, with_year = FALSE) Quarter. *quarter(dt)*
	semester(x, with_year = FALSE) Semester. *semester(dt)*
	am(x) Is it in the am? *am(dt)* **pm**(x) Is it in the pm? *pm(dt)*
	dst(x) Is it daylight savings? *dst(d)*
	leap_year(x) Is it a leap year? *leap_year(d)*
	update(object, ..., simple = FALSE) *update(dt, mday = 2, hour = 1)*

Round Date-times

floor_date(x, unit = "second") Round down to nearest unit. *floor_date(dt, unit = "month")*

round_date(x, unit = "second") Round to nearest unit. *round_date(dt, unit = "month")*

ceiling_date(x, unit = "second", change_on_boundary = NULL) Round up to nearest unit. *ceiling_date(dt, unit = "month")*

rollback(dates, roll_to_first = FALSE, preserve_hms = TRUE) Roll back to last day of previous month. *rollback(dt)*

Stamp Date-times

stamp() Derive a template from an example string and return a new function that will apply the template to date-times. Also **stamp_date**() and **stamp_time**().

1. Derive a template, create a function
   sf <- stamp("Created Sunday, Jan 17, 1999 3:34")
2. Apply the template to dates
   sf(ymd("2010-04-05"))
   ## [1] "Created Monday, Apr 05, 2010 00:00"

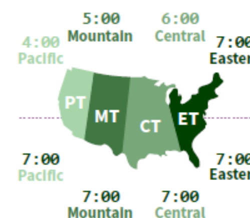
Tip: use a date with day > 12

Time Zones

R recognizes ~600 time zones. Each encodes the time zone, Daylight Savings Time, and historical calendar variations for an area. R assigns *one* time zone per vector.

Use the **UTC** time zone to avoid Daylight Savings.

OlsonNames() Returns a list of valid time zone names. *OlsonNames()*



with_tz(time, tzone = "") Get the **same date-time** in a new time zone (a new clock time). *with_tz(dt, "US/Pacific")*

force_tz(time, tzone = "") Get the **same clock time** in a new time zone (a new date-time). *force_tz(dt, "US/Pacific")*

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at lubridate.tidyverse.org • lubridate 1.6.0 • Updated: 2017-12

Pacote **lubridate**

```
library(lubridate)
x <- Sys.time()
print(x)
```

```
## [1] "2020-10-19 19:01:55 -03"
```

```
date(x)
```

```
## [1] "2020-10-19"
```

```
year(x)
```

```
## [1] 2020
```

Pacote **lubridate**

```
isoyear(x)
```

```
## [1] 2020
```

```
epiyear(x)
```

```
## [1] 2020
```

```
month(x, abbr = TRUE)
```

```
## [1] 10
```

```
second(x)
```

```
## [1] 55.92229
```

Pacote **lubridate**

```
leap_year(x)
```

```
## [1] TRUE
```

```
quarter(x)
```

```
## [1] 4
```

```
semester(x)
```

```
## [1] 2
```

```
dst(x)
```

```
## [1] FALSE
```

Pacote **lubridate**

```
floor_date(x, unit = "day")
```

```
## [1] "2020-10-19 -03"
```

```
ceiling_date(x, unit = "hour")
```

```
## [1] "2020-10-19 20:00:00 -03"
```

```
round_date(x, unit = "month")
```

```
## [1] "2020-11-01 -03"
```

Pacote `lubridate`

```
leap <- ymd("2019-03-01")
print(leap)
```

```
## [1] "2019-03-01"
```

```
leap + years(1)
```

```
## [1] "2020-03-01"
```

```
leap + dyears(1)
```

```
## [1] "2020-02-29 06:00:00 UTC"
```

```
leap + 365
```

```
## [1] "2020-02-29"
```

Pacote **lubridate**

```
dseconds(1)
```

```
## [1] "1s"
```

```
dyears(40)
```

```
## [1] "1262304000s (~40 years)"
```

```
dweeks(1)
```

```
## [1] "604800s (~1 weeks)"
```

```
dminutes(c(1:4))
```

```
## [1] "60s (~1 minutes)"  "120s (~2 minutes)" "180s (~3 minutes)"
## [4] "240s (~4 minutes)"
```