| Please choose a lesson, or type 0 to return to course menu.

1: Basic Building Blocks 2: Workspace and Files 3: Sequences of Numbers

4: Vectors 5: Missing Values 6: Subsetting Vectors

7: Matrices and Data Frames 8: Logic 9: Functions

10: lapply and sapply 11: vapply and tapply 12: Looking at Data

13: Simulation 14: Dates and Times 15: Base Graphics

Selection: 14

| | 0%

| R has a special way of representing dates and times, which can be helpful if you're working with data

| that show how something changes over time (i.e. time-series data) or if your data contain some other

| temporal information, like dates of birth.

...

|=== | 3%

| Dates are represented by the 'Date' class and times are represented by the 'POSIXct' and 'POSIXlt'

| classes. Internally, dates are stored as the number of days since 1970-01-01 and times are stored as

| either the number of seconds since 1970-01-01 (for 'POSIXct') or a list of seconds, minutes, hours,

| etc. (for 'POSIXlt').

...

|===== | 6%

| Let's start by using d1 <- Sys.Date() to get the current date and store it in the variable d1. (That's

| the letter 'd' and the number 1.)

> d1 <- Sys.Date()

| You nailed it! Good job!

|======== | 9%

| Use the class() function to confirm d1 is a Date object.

> class(d1)

[1] "Date"

| Excellent work!

|=========== | 11%

| We can use the unclass() function to see what d1 looks like internally. Try it out.

> unclass(d1)

[1] 16837

| Perseverance, that's the answer.

|============== | 14%

| That's the exact number of days since 1970-01-01!

...

|================ | 17%

| However, if you print d1 to the console, you'll get today's date -- YEAR-MONTH-DAY. Give it a try.

> d1

[1] "2016-02-06"

| You are amazing!

|=================== | 20%

| What if we need to reference a date prior to 1970-01-01? Create a variable d2 containing

| as.Date("1969-01-01").

> d2 <- as.Date("1969-01-01")

| That's a job well done!

|====================== | 23%

| Now use unclass() again to see what d2 looks like internally.

> unclass(d2)

[1] -365

| Great job!

|========================= | 26%

| As you may have anticipated, you get a negative number. In this case, it's -365, since 1969-01-01 is

| exactly one calendar year (i.e. 365 days) BEFORE 1970-01-01.

...

|=========================== | 29%

| Now, let's take a look at how R stores times. You can access the current date and time using the

| Sys.time() function with no arguments. Do this and store the result in a variable called t1.

> t1 <- Sys.time()

| Keep working like that and you'll get there!

|============================== | 31%

| View the contents of t1.

> t1

[1] "2016-02-06 11:31:18 CET"

| Keep working like that and you'll get there!

|================================= | 34%

| And check the class() of t1.

> class(t1)

[1] "POSIXct" "POSIXt"

| You're the best!

|==================================== | 37%

| As mentioned earlier, POSIXct is just one of two ways that R represents time information. (You can

| ignore the second value above, POSIXt, which just functions as a common language between POSIXct and

| POSIXlt.) Use unclass() to see what t1 looks like internally -- the (large) number of seconds since the

| beginning of 1970.

> unclass(t1)

[1] 1454754679

| You are doing so well!

|====================================== | 40%

| By default, Sys.time() returns an object of class POSIXct, but we can coerce the result to POSIXlt with

| as.POSIXlt(Sys.time()). Give it a try and store the result in t2.

> t2 <- as.POSIXlt(Sys.time())

| Great job!

|========================================= | 43%

| Check the class of t2.

> class(t2)

[1] "POSIXlt" "POSIXt"

| Your dedication is inspiring!

|============================================ | 46%

| Now view its contents.

> t2

[1] "2016-02-06 11:32:50 CET"

| That's a job well done!

|=============================================== | 49%

| The printed format of t2 is identical to that of t1. Now unclass() t2 to see how it is different

| internally.

> unclass(t2)

$sec

[1] 50.39486

$min

[1] 32

$hour

[1] 11

$mday

[1] 6

$mon

[1] 1

$year

[1] 116

$wday

[1] 6

$yday

[1] 36

$isdst

[1] 0

$zone

[1] "CET"

$gmtoff

[1] 3600

attr(,"tzone")

[1] "" "CET" "CEST"

| Perseverance, that's the answer.

|================================================= | 51%

| t2, like all POSIXlt objects, is just a list of values that make up the date and time. Use

| str(unclass(t2)) to have a more compact view.

> str(unclass(t2))

List of 11

$ sec : num 50.4

$ min : int 32

$ hour : int 11

$ mday : int 6

$ mon : int 1

$ year : int 116

$ wday : int 6

$ yday : int 36

$ isdst : int 0

$ zone : chr "CET"

$ gmtoff: int 3600

- attr(\*, "tzone")= chr [1:3] "" "CET" "CEST"

| That's correct!

|==================================================== | 54%

| If, for example, we want just the minutes from the time stored in t2, we can access them with t2$min.

| Give it a try.

> t2$min

[1] 32

| You got it!

|======================================================= | 57%

| Now that we have explored all three types of date and time objects, let's look at a few functions that

| extract useful information from any of these objects -- weekdays(), months(), and quarters().

...

|========================================================== | 60%

| The weekdays() function will return the day of week from any date or time object. Try it out on d1,

| which is the Date object that contains today's date.

> weekdays(d1)

[1] "sábado"

| You got it!

|============================================================ | 63%

| The months() function also works on any date or time object. Try it on t1, which is the POSIXct object

| that contains the current time (well, it was the current time when you created it).

> months(t1)

[1] "febrero"

| Excellent work!

|=============================================================== | 66%

| The quarters() function returns the quarter of the year (Q1-Q4) from any date or time object. Try it on

| t2, which is the POSIXlt object that contains the time at which you created it.

> quarters(t2)

[1] "Q1"

| All that hard work is paying off!

|================================================================== | 69%

| Often, the dates and times in a dataset will be in a format that R does not recognize. The strptime()

| function can be helpful in this situation.

...

|===================================================================== | 71%

| strptime() converts character vectors to POSIXlt. In that sense, it is similar to as.POSIXlt(), except

| that the input doesn't have to be in a particular format (YYYY-MM-DD).

...

|======================================================================= | 74%

| To see how it works, store the following character string in a variable called t3: "October 17, 1986

| 08:24" (with the quotes).

> t3 <- "October 17, 1986 08:24"

| You nailed it! Good job!

|========================================================================== | 77%

| Now, use strptime(t3, "%B %d, %Y %H:%M") to help R convert our date/time object to a format that it

| understands. Assign the result to a new variable called t4. (You should pull up the documentation for

| strptime() if you'd like to know more about how it works.)

> ?strptime

> t4 <- strptime(t3, "%B %d, %Y %H:%M")

| All that practice is paying off!

|============================================================================= | 80%

| Print the contents of t4.

> t4

[1] NA

| You are doing so well!

|================================================================================ | 83%

| That's the format we've come to expect. Now, let's check its class().

> class(t4)

[1] "POSIXlt" "POSIXt"

| Great job!

|================================================================================== | 86%

| Finally, there are a number of operations that you can perform on dates and times, including arithmetic

| operations (+ and -) and comparisons (<, ==, etc.)

...

|===================================================================================== | 89%

| The variable t1 contains the time at which you created it (recall you used Sys.time()). Confirm that some

| time has passed since you created t1 by using the 'greater than' operator to compare it to the current

| time: Sys.time() > t1

> Sys.time() > t1

[1] TRUE

| Nice work!

|======================================================================================== | 91%

| So we know that some time has passed, but how much? Try subtracting t1 from the current time using

| Sys.time() - t1. Don't forget the parentheses at the end of Sys.time(), since it is a function.

> Sys.time() - t1

Time difference of 10.38714 mins

| Nice work!

|=========================================================================================== | 94%

| The same line of thinking applies to addition and the other comparison operators. If you want more

| control over the units when finding the above difference in times, you can use difftime(), which allows

| you to specify a 'units' parameter.

...

|============================================================================================= | 97%

| Use difftime(Sys.time(), t1, units = 'days') to find the amount of time in DAYS that has passed since you

| created t1.

> difftime(Sys.time(), t1, units = 'days')

Time difference of 0.007657493 days

| You are quite good my friend!

|================================================================================================| 100%

| In this lesson, you learned how to work with dates and times in R. While it is important to understand

| the basics, if you find yourself working with dates and times often, you may want to check out the

| lubridate package by Hadley Wickham.

...