```
Selection: 14

  |
|   0%

| R has a special way of representing dates and times, which can be helpful i
f you're working with
| data that show how something changes over time (i.e. time-series data) or i
f 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 1970-01-0
1 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()

| Keep up the great work!

  |========
|   8%
| Use the class() function to confirm d1 is a Date object.

> class(d1)
[1] "Date"

| Keep up the great work!

  |==========
|  11%
| We can use the unclass() function to see what d1 looks like internally. Try
it out.

> unclass(d1)
[1] 18368

| You are doing so well!

  |=============
|  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-MO
NTH-DAY. Give it a try.
```

```
> d1
[1] "2020-04-16"

| You are doing so well!

  |==================
|  19%
| 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")

| You're the best!

  |=====================
|  22%
| Now use unclass() again to see what d2 looks like internally.

> unclass(d2)
[1] -365

| You got it!

  |=======================
|  25%
| 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.

...

  |==========================
|  28%
| Now, let's take a look at how R stores times. You can access the current da
te and time using the
| Sys.time() function with no arguments. Do this and store the result in a va
riable called t1.

> ti<- Sys.time()

| Almost! Try again. Or, type info() for more options.

| t1 <- Sys.time() will store the current date and time in a variable called
t1.

> t1<- Sys.time()

| You're the best!

  |============================
|  31%
| View the contents of t1.

> t1
[1] "2020-04-16 22:39:16 IST"

| You got it right!

  |==============================
|  33%
| And check the class() of t1.
```

```
> class(t1)
[1] "POSIXct" "POSIXt"

| Excellent job!

  |=================================
|  36%
| As mentioned earlier, POSIXct is just one of two ways that R represents tim
e information. (You can
| ignore the second value above, POSIXt, which just functions as a common lan
guage 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] 1587056957

| Your dedication is inspiring!

  |==================================
|  39%
| By default, Sys.time() returns an object of class POSIXct, but we can coerc
e the result to POSIXlt
| with as.POSIXlt(Sys.time()). Give it a try and store the result in t2.

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

| You nailed it! Good job!

  |====================================
|  42%
| Check the class of t2.

> t2
[1] "2020-04-16 22:41:35 IST"

| Nice try, but that's not exactly what I was hoping for. Try again. Or, type
info() for more options.

| Type class(t2) to view its class.

> class(t2)
[1] "POSIXlt" "POSIXt"

| That's the answer I was looking for.

  |=====================================
|  44%
| Now view its contents.

> t2
[1] "2020-04-16 22:41:35 IST"

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

  |=======================================
|  47%
| The printed format of t2 is identical to that of t1. Now unclass() t2 to se
e how it is different
| internally.

> unclass(t2)
$sec
```

```
[1] 35.57194

$min
[1] 41

$hour
[1] 22

$mday
[1] 16

$mon
[1] 3

$year
[1] 120

$wday
[1] 4

$yday
[1] 106

$isdst
[1] 0

$zone
[1] "IST"

$gmtoff
[1] 19800

attr(,"tzone")
[1] ""      "IST"   "+0630"

| All that hard work is paying off!

  |=============================================
|  50%
| t2, like all POSIXlt objects, is just a list of values that make up the dat
e and time. Use
| str(unclass(t2)) to have a more compact view.

> str(unclass(t2))
List of 11
 $ sec   : num 35.6
 $ min   : int 41
 $ hour  : int 22
 $ mday  : int 16
 $ mon   : int 3
 $ year  : int 120
 $ wday  : int 4
 $ yday  : int 106
 $ isdst : int 0
 $ zone  : chr "IST"
 $ gmtoff: int 19800
 - attr(*, "tzone")= chr [1:3] "" "IST" "+0630"

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

  |=============================================
|  53%
| If, for example, we want just the minutes from the time stored in t2, we ca
n access them with
```

```
| t2$min. Give it a try.

> t2$min
[1] 41

| You are amazing!

  |=======================================================
|  56%
| Now that we have explored all three types of date and time objects, let's l
ook at a few functions
| that extract useful information from any of these objects -- weekdays(), mo
nths(), and quarters().

...

  |=========================================================
|  58%
| The weekdays() function will return the day of week from any date or time o
bject. Try it out on d1,
| which is the Date object that contains today's date.

> weekdays(d1)
[1] "Thursday"

| You got it right!

  |===========================================================
|  61%
| 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 y
ou created it).

> months(t1)
[1] "April"

| You nailed it! Good job!

  |=============================================================
|  64%
| The quarters() function returns the quarter of the year (Q1-Q4) from any da
te or time object. Try it
| on t2, which is the POSIXlt object that contains the time at which you crea
ted it.

> quarters(t2)
[1] "Q2"

| All that hard work is paying off!

  |===============================================================
|  67%
| 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.

...

  |=================================================================
|  69%
| strptime() converts character vectors to POSIXlt. In that sense, it is simi
lar to as.POSIXlt(),
```

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

...

  |=======================================================================
|  72%
| To see how it works, store the following character string in a variable cal
led t3: "October 17, 1986
| 08:24" (with the quotes).

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

| That's a job well done!

  |=======================================================================
|  75%
| Now, use strptime(t3, "%B %d, %Y %H:%M") to help R convert our date/time ob
ject to a format that it
| understands. Assign the result to a new variable called t4. (You should pul
l up the documentation
| for strptime() if you'd like to know more about how it works.)

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

| You got it right!

  |=======================================================================
|  78%
| Print the contents of t4.

> t4
[1] "1986-10-17 08:24:00 IST"

| Keep up the great work!

  |=======================================================================
=                    |  81%
| That's the format we've come to expect. Now, let's check its class().

> class(t4)
[1] "POSIXlt" "POSIXt"

| Keep up the great work!

  |=======================================================================
====                 |  83%
| Finally, there are a number of operations that you can perform on dates and
times, including
| arithmetic operations (+ and -) and comparisons (<, ==, etc.)

...

  |=======================================================================
======               |  86%
| 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' opera
tor to compare it to the
| current time: Sys.time() > t1

> Sys.time() > t1
[1] TRUE
```

```
| Nice work!

  |=======================================================================
=========          |  89%
| 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(), sin
ce it is a function.

> Sys.time() - t1
Time difference of 13.66982 mins

| You are doing so well!

  |=======================================================================
===========        |  92%
| The same line of thinking applies to addition and the other comparison oper
ators. 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.

...

  |=======================================================================
==============     |  94%
| 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 0.0100779 days

| You are doing so well!

  |=======================================================================
================   |  97%
| 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 of
ten, you may want to
| check out the lubridate package by Hadley Wickham.

...

  |=======================================================================
==================| 100%
| Would you like to receive credit for completing this course on Coursera.org
?
```