```
#=============================================================================================
# Part 2: Indexing with logicals and objects
#=============================================================================================

#---------------------------------------------------------------------------------------------
# A primer on logicals.
#---------------------------------------------------------------------------------------------

# >> First, a list of logical operators:
#  <      Less than.
#  >      Greater than.
#  <=     Less than or equal to.
#  >=     Greater than or equal to.
#  ==     Is exactly equal to (not just for numbers, e.g., "yes"=="yes" = TRUE).
#  !      Negation, or 'bang' operator. Often translatable as "not".
#  !=     Not equal to.
#  &      And. Used for doing multiple logical tests, e.g., (sky == "blue" & sky ==
#         "cloudy"). Returns TRUE only if both conditions are true. Analogous to the
#         'intersection' in set theory.
#  |      Or. Used for doing multiple logical tests, e.g., (sky == "blue" | sky ==
#         "cloudy"). Returns TRUE if either condition is true, or if both are true.
#         Returns true if either condition is true, or if both are true. Analogous to the
#         'union' in set theory.
#  xor()  The 'either' function. Used for doing multiple logical tests. Returns TRUE only
#         when either condition is true, but returns FALSE if both conditions are true.
#         This is the 'intersection complement' in the 'union' in set theory.
#  &&, || This type is used for control flow in "if" statements. We'll get to those later.

# The simplest use of logicals are element-wise tests. When a logical is directed at a
# group of elements, each element is tested, and a TRUE or FALSE is returned for each one,
# resulting in a logical vector. The length of the answer therefore matches the length of
# the group tested.

# << A few examples >> ------------------------------------------------------------------

#--Test whether each name of 'x' is not equal to "chol".
names (x) != "chol"
# The last three names are not equal to "chol".

#--Test whether each element of 'x' is greater than 80.
x > 80

#--Text whether each element of 'x' is less than 70 or greater than 200.
x < 70 | x > 200

# Notice there are no parentheses, brackets, or commas here. The & and | signs separate
# multiple tests.

#--Since you can index with logicals, return the elements of 'x' that are less than 70 or
# greater than 200 by inserting the logical-test series above inside x[ ].
x [x < 70 | x > 200]

# << The which() function >> ------------------------------------------------------------

# An *essential function*, which() looks at TRUE/FALSE vectors, and asks "which positions
# contain elements that are TRUE?". It returns the integer positions of TRUE elements.
which (c(TRUE, FALSE, FALSE, TRUE))
class (which (c(TRUE, FALSE, FALSE, TRUE)))

# which() will therefore return the positions where the answer to your logical test is
# TRUE.

#--Make a vector 10:20. Return the positions of v greater than 15.
v <- 10:20
```

```
which (v > 15)

#--Return the positions in the vector of names of our vector 'x' not equal to "chol".
which (names(x) != "chol")

# Now, remember how we couldn't use negative indexing on names? Let's try that again,
# using which() to turn names into numbers.

#--Return all of the elements of 'x' EXCLUDING the one named "chol", combining which()
# with negative indexing.
x [ - which (names (x) == "chol") ]

# Let's break that up, just to drill these concepts in. First, look at names(x).
names(x)
# Test whether names(x) are equal to "chol".
names(x) == "chol"
# Ask which() names(x) positions are qual to "chol"
which (names(x) == "chol")
# The answer is 1. So just as you can negatively index 1 to remove it...
x [-1]
# you can insert the code that returns the positions of passed tests, and remove them.
x [ - which (names (x) == "chol") ]

#--Use which() and length() to find out how many elements of 'v' are greater than 13 and
# not equal to 17.
length (which (v > 13 & v != 17))
# Take a look at 'v' to verify your result.
v

#--See what happens when you don't use which() for the above formula.
length (v > 13 & v != 17)
# Make sure you understand that answer. Take a look at the results of the logical test
# alone, and think about what you are asking the length() of.
v > 13 & v != 17
```