Video 8: Subsetting Lists

Stats 102A

Miles Chen, PhD

Subsetting Lists

Subsetting Lists

Subsetting a list works in the same way as subsetting an atomic vector.

Important: Using a single square bracket `[` will always return a list.

Using a double square bracket `[[` or dollar-sign `$`, as described next, let you pull out the components of the list.

Subsetting operators

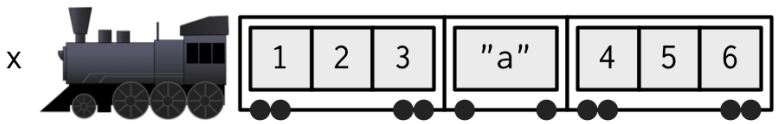
There subsetting operators: `[[ ]]` and `$` are similar to `[ ]`, except it can only return a single object and it allows you to pull pieces out of a list. `$` is a useful shorthand for `[[ ]]` combined with character subsetting.

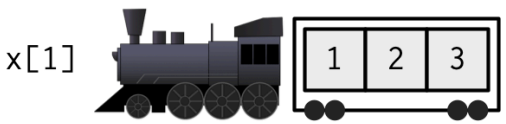
You need `[[ ]]` when working with lists. This is because when `[ ]` is applied to a list it always returns a list: it never gives you the contents of the list. To get the contents, you need `[[ ]]`:

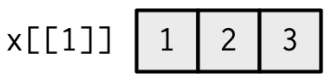
“If list `x` is a train carrying objects, then `x[[5]]` is the object in car 5; `x[4:6]` is a train of cars 4-6.” - @RLangTip

From Advanced R, by Hadley Wickham

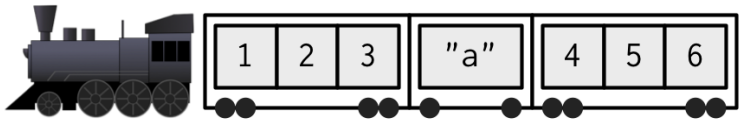
```r
x <- list(1:3, "a", 4:6)
```



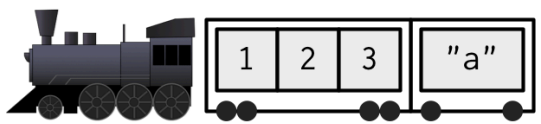




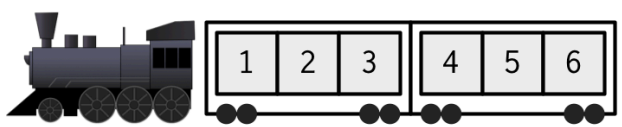
x



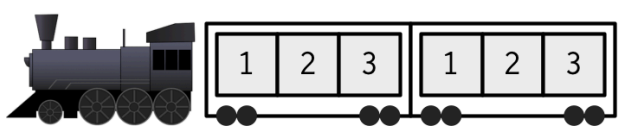
x[1:2]



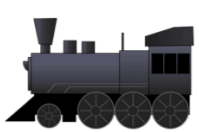
x[-2]



x[c(1,1)]



x[0]



Double square brackets

Because it can return only a single object, you must use `[[ ]]` with either a single positive integer or a single string

```r
x <- list(a = 1:3, b = "a", c = 4:6)
x
```

```
$a
[1] 1 2 3

$b
[1] "a"

$c
[1] 4 5 6
```

```r
x[[1]]
```

```
[1] 1 2 3
```

```r
x[["a"]]
```

```
[1] 1 2 3
```

Single vs double square brackets

```
1 x[c("a", "b", "b")] # can use single brackets with a vector of mult
```

```
$a
[1] 1 2 3

$b
[1] "a"

$b
[1] "a"
```

```
1 x[[ c("a", "b") ]] # this does not work
```

```
Error in x[[c("a", "b")]]: subscript out of bounds
```

Sidenote: Recursive Subsetting

Putting a vector of multiple elements in double square brackets performs recursive subsetting. `x[[c("a","b")]]` is actually equivalent to `x[["a"]][["b"]]` which only works if `x` is a list with element `a`, and `a` itself has an element inside it called `b`.

```
1 x <- list(a = list(a = "a", b = "b"), b = 1:3)
2 str(x)
```

```
List of 2
 $ a:List of 2
  ..$ a: chr "a"
  ..$ b: chr "b"
 $ b: int [1:3] 1 2 3
```

```
1 x[["a"]][["b"]]
```

```
[1] "b"
```

```
1 x[[c("a","b")]]
```

```
[1] "b"
```

Another Example

```r
d <- list(
  a = c(1, 2, 3),
  b = c(TRUE, TRUE, FALSE),
  c = c("a")
  )
str(d)
```

```
List of 3
 $ a: num [1:3] 1 2 3
 $ b: logi [1:3] TRUE TRUE FALSE
 $ c: chr "a"
```

Single square bracket vs double square bracket

```
d[1] # single square bracket returns a list containing the first el
```

```
$a
[1] 1 2 3
```

```
typeof(d[1])
```

```
[1] "list"
```

```
d[[1]] # double square bracket returns the contents of the first el
```

```
[1] 1 2 3
```

```
typeof(d[[1]])
```

```
[1] "double"
```

Let’s make a new list

```r
l1 <- list(
  1:8,
  letters[1:4],
  5:1
  )  # The list has three elements, but they are unnamed
str(l1)
```

```
List of 3
 $ : int [1:8] 1 2 3 4 5 6 7 8
 $ : chr [1:4] "a" "b" "c" "d"
 $ : int [1:5] 5 4 3 2 1
```

Single square bracket vs Double square bracket

```
1 l1[1] # this is a list. returns the first train car
```

```
[[1]]
[1] 1 2 3 4 5 6 7 8
```

```
1 l1[[1]] # this is a vector. the contents of the first train car
```

```
[1] 1 2 3 4 5 6 7 8
```

List subsetting

```
l1[1][2]   # l1[1] is a list of one item. It has no second element
```

```
[[1]]
NULL
```

```
l1[[1]][2] # l1[[1]] is the integer vector 1:8. The second element
```

```
[1] 2
```

```
l1[[c(1,2)]] # Recursive subsetting: l1[[c(1,2)]] is equal to l1[[1
```

```
[1] 2
```

A list inside a list (A train inside a train car)

```r
# l2 has 3 elements: the first is the list l1
# the second and third elements are vectors
l2 <- list( l1, c(10, 20, 30), LETTERS[4:9])
str(l2)
```

```
List of 3
 $ :List of 3
  ..$ : int [1:8] 1 2 3 4 5 6 7 8
  ..$ : chr [1:4] "a" "b" "c" "d"
  ..$ : int [1:5] 5 4 3 2 1
 $ : num [1:3] 10 20 30
 $ : chr [1:6] "D" "E" "F" "G" ...
```

Single square bracket returns the list (train car with the train inside)

```
# subsetting with a single square bracket returns a list of one ele
# which itself contains the list l1
l2[1]
```

```
[[1]]
[[1]][[1]]
[1] 1 2 3 4 5 6 7 8

[[1]][[2]]
[1] "a" "b" "c" "d"

[[1]][[3]]
[1] 5 4 3 2 1
```

Single square bracket returns the list (train car with the train inside)

```
1 # str shows that it is a list in a list
2 str(l2[1])
```

```
List of 1
 $ :List of 3
  ..$ : int [1:8] 1 2 3 4 5 6 7 8
  ..$ : chr [1:4] "a" "b" "c" "d"
  ..$ : int [1:5] 5 4 3 2 1
```

Double square bracket returns the contents (the actual train inside)

```
# double square bracket returns the contents of the first element
# which is the list l1 (just the listt)
l2[[1]]
```

```
[[1]]
[1] 1 2 3 4 5 6 7 8

[[2]]
[1] "a" "b" "c" "d"

[[3]]
[1] 5 4 3 2 1
```

Double square bracket returns the contents (the actual train inside)

```
# str reveals we have just the list, not a list in a list
str(l2[[1]])
```

```
List of 3
 $ : int [1:8] 1 2 3 4 5 6 7 8
 $ : chr [1:4] "a" "b" "c" "d"
 $ : int [1:5] 5 4 3 2 1
```

Pay attention to the differences

```
1  l2[[1]][1]
```

```
[[1]]
[1] 1 2 3 4 5 6 7 8
```

```
1  l2[[1]][1][2]
```

```
[[1]]
NULL
```

```
1  l2[[1]][[1]]
```

```
[1] 1 2 3 4 5 6 7 8
```

```
1  l2[[1]][[1]][2]
```

```
[1] 2
```

Double square brackets on atomic vectors

Double square brackets with atomic vectors behave similarly to the use of single square brackets with a few key differences, particularly in handling out-of-bounds indexes.

```
x <- c("a" = 1, "b" = 2, "c" = 3)
x[1] # single square brackets preserve names
```

```
a
1
```

```
x[[1]] # double square brackets drop names
```

```
[1] 1
```

Double square brackets on atomic vectors

```
x <- 1:3
x[5] # single square brackets return NA for out-of-bounds index
```

```
[1] NA
```

```
x[[5]] # double square brackets return error for out-of-bounds inde
```

```
Error in x[[5]]: subscript out of bounds
```

```
x[NULL] # single square brackets return length 0 vectors for 0 or N
```

```
integer(0)
```

```
x[[NULL]] # double square brackets return error for 0 or NULL
```

```
Error in x[[NULL]]: attempt to select less than one element in get1index
```

Single vs Double brackets for OOB and NULL indices

Similar rules for out-of-bounds indexes happen with single vs double square brackets with lists. Let's take a look at l1 again.

```
          1  l1
```

```
[[1]]
[1] 1 2 3 4 5 6 7 8

[[2]]
[1] "a" "b" "c" "d"

[[3]]
[1] 5 4 3 2 1
```

Single vs Double brackets for OOB and NULL indices

```
1 l1[4] # out of bounds returns a train car with NULL inside
```

```
[[1]]
NULL
```

```
1 l1[[4]] # common error in for loops
```

```
Error in l1[[4]]: subscript out of bounds
```

```
1 l1[NULL] # NULL or 0 returns no train cars
```

```
list()
```

```
1 l1[[NULL]]
```

```
Error in l1[[NULL]]: attempt to select less than one element in get1index
```