Lecture 11 - Loops - `for`, `while`, `repeat`

Packages

```
# none
```

What are loops?

- A loop is a sequence of statements carried out several times in succession.
- There are three major types of loops:
    1. `for` loops
    2. `while` loops
    3. `repeat` loops
- `for` loops are by far the most commonly used type of looping mechanism, but `while` and `repeat` loops are better for certain situations.

`for` Loops

`for` loop syntax

- The `for` loop iterates through each element of a vector or list and performs a task at each iteration
- The `for` loop completes execution after iterating through all elements of a vector or list
- `for` loop syntax

```
for (iterator in vector/list) {
  for loop body
}
```

- `for`
  - A `for` loop is initiated using the `for` keyword
- `iterator`
  - The iterator is a variable that iterates through each element of the vector
- `in`
  - `in` is a special keyword that tells `R` we want to iterate through the elements "in" the vector or list
- `vector/list`
  - The `for` loop will use the `iterator` to iterate through each element of a vector or list. This includes columns of data frames since data frames are structured lists!
- `for` loop body
  - At each iteration, the code in the `for` loop body is executed

`for` Loop Examples

- Most often, `for` loops are used to iterate through a sequence of consecutive integers.
- A vector of consecutive integers is created using a colon `:`

```
# create a vector of consecutive integers
1:10
```

```
1 · 2 · 3 · 4 · 5 · 6 · 7 · 8 · 9 · 10
```

- The iterator `i` iterates through each entry in the vector `1:10` and performs the given `print()` task

```
# simple for loop
for (i in 1:10) {
  print(i)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

- We can use the function `seq()` to iterate through other sequence patterns

```
# create a vector from 1 to 10 by 2
vec <- seq(1, 10, 2)
vec
```

```
1 · 3 · 5 · 7 · 9
```

```
# loop through each entry in the variable vec
for (j in vec) {
  print(j)
}
```

```
[1] 1
[1] 3
[1] 5
[1] 7
[1] 9
```

- We can loop through character vectors as well!

```
vec <- c("Baby", "Yoda", "is", "not", "really", "baby", "Yoda")
for (ii in vec) {
```

```
for (ii in vec) {
  print(ii)
}
```

```
[1] "Baby"
[1] "Yoda"
[1] "is"
[1] "not"
[1] "really"
[1] "baby"
[1] "Yoda"
```

- Remember, data frames are structured lists, where each column is a list item
- Therefore, we can loop through each column

```
# load sample dataset
data(mtcars)
head(mtcars)
```

A data.frame: 6 × 11

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	c
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<d
Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	
Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	
Hornet											

```
# Loop through columns and calculate means
for (i in mtcars) {
  print(mean(i))
}
```

```
[1] 20.09062
[1] 6.1875
[1] 230.7219
[1] 146.6875
[1] 3.596563
[1] 3.21725
[1] 17.84875
[1] 0.4375
[1] 0.40625
[1] 3.6875
[1] 2.8125
```

- But what if we are only interested in the numeric variables for their averages?

```
# only select mpg, hp, and wt
head(mtcars[, c("mpg", "hp", "wt")])
```

A data.frame: 6 × 3

	mpg	hp	wt
	<dbl>	<dbl>	<dbl>
Mazda RX4	21.0	110	2.620
Mazda RX4 Wag	21.0	110	2.875
Datsun 710	22.8	93	2.320
Hornet 4 Drive	21.4	110	3.215
Hornet Sportabout	18.7	175	3.440
Valiant	18.1	105	3.460

```
# calculate means using a loop
for (cols in mtcars[, c("mpg", "hp", "wt")]) {
  print(mean(cols))
}
```

```
[1] 20.09062
[1] 146.6875
[1] 3.21725
```

`for` Loops and Populating Empty Vectors

- The previous example calculates the mean of three variables in a data frame
- However, what if we want to store these averages for future use?

- We can store the averages in an empty vector as we iterate through our loop!
- The script below creates an empty vector

```
# create an empty vector
mtcars_means <- vector(length = 3)
mtcars_means
```

FALSE · FALSE · FALSE

- Numeric indexing and Boolean-based subsetting not only allows us to access entries of a vector...
  - We can also overwrite these entries

```
# access the second entry of this vector
mtcars_means[2]
```

FALSE

```
# overwrite the second entry of this vector
mtcars_means[2] <- 3.14
mtcars_means
```

0 · 3.14 · 0

- Let's use this concept to store our means

```
1:ncol(mtcars_subset)
```

1 · 2 · 3

```
# subset the columns in which we are interested
mtcars_subset <- mtcars[, c("mpg", "hp", "wt")]

# create an empty vector
mtcars_means <- vector(length = ncol(mtcars_subset))


# loop through the columns
for (cols in 1:ncol(mtcars_subset)) {

  # store the means
  mtcars_means[cols] <- mean(mtcars_subset[, cols])

}
```

```
mtcars_means
```

```
20.090625 · 146.6875 · 3.21725
```

- Another approach to storing values is to use `c()` to concatenate additional values to a vector

```
# create an empty vector using c()
mtcars_means <- c()
mtcars_means
```

```
NULL
```

```
# use c() to concatenate a value to the empty vector
mtcars_means <- c(mtcars_means, 3.14)
mtcars_means
```

```
3.14
```

```
# again, use c() to concatenate a value to the empty vector
mtcars_means <- c(mtcars_means, 2.7)
mtcars_means
```

```
3.14 · 2.7
```

- Let's use this concept to store our means

```
# subset the columns in which we are interested
mtcars_subset <- mtcars[, c("mpg", "hp", "wt")]

# create an empty vector
mtcars_means <- c()

# loop through the columns
for (cols in 1:ncol(mtcars_subset)) {

  # store the means using concatenation
  mtcars_means <- c(mtcars_means, mean(mtcars_subset[, cols]))
  print(mtcars_means)

}
```

```
[1] 20.09062
[1] 20.09062 146.68750
```

```
[1]  20.09062 146.68750
[1]  20.09062 146.68750   3.21725
```

```
mtcars_means
```

```
20.090625 · 146.6875 · 3.21725
```

- The above approach is useful if the number of items being stored is unknown
- For example, what if we only want to store means that are less than 100?

```
# subset the columns in which we are interested
mtcars_subset <- mtcars[, c("mpg", "hp", "wt")]

# create an empty vector
mtcars_means <- c()

# loop through the columns
for (cols in 1:ncol(mtcars_subset)) {

  # calculate mean
  m <- mean(mtcars_subset[, cols])

  if (m < 100) {
    # store the means using concatenation
    mtcars_means <- c(mtcars_means, m)
  }

}
```

```
mtcars_means
```

```
20.090625 · 3.21725
```

Nested `for` loops

- We can also nest loops (loops within loops)
- This is typically inefficient and can be alternatively performed using vectorization!

```
# what is it doing on the inside?
for (jj in 1:ncol(mtcars_subset)) {
```

```
  for (ii in 1:length(mtcars_subset[,jj])) {

    print(paste(ii, jj))

  } # end inner loop
} # end outer loop
```

```
[1] "1 1"
[1] "2 1"
[1] "3 1"
[1] "4 1"
[1] "5 1"
[1] "6 1"
[1] "7 1"
[1] "8 1"
[1] "9 1"
[1] "10 1"
[1] "11 1"
[1] "12 1"
[1] "13 1"
[1] "14 1"
[1] "15 1"
[1] "16 1"
[1] "17 1"
[1] "18 1"
[1] "19 1"
[1] "20 1"
[1] "21 1"
[1] "22 1"
[1] "23 1"
[1] "24 1"
[1] "25 1"
[1] "26 1"
[1] "27 1"
[1] "28 1"
[1] "29 1"
[1] "30 1"
[1] "31 1"
[1] "32 1"
[1] "1 2"
[1] "2 2"
[1] "3 2"
[1] "4 2"
[1] "5 2"
[1] "6 2"
[1] "7 2"
[1] "8 2"
[1] "9 2"
[1] "10 2"
[1] "11 2"
[1] "12 2"
[1] "13 2"
[1] "14 2"
[1] "15 2"
[1] "16 2"
[1] "17 2"
[1] "18 2"
[1] "19 2"
```

```
[1] "20 2"
[1] "21 2"
[1] "22 2"
[1] "23 2"
[1] "24 2"
[1] "25 2"
[1] "26 2"
```

`while` loops

`while` loop syntax

- The `while` loop repeatedly iterates "while" a certain condition is true.
- `while` loops are very useful for stopping an algorithm when convergence is met.
- `while` loop syntax

```
while (boolean expression) {
while loop body
}
```

- `while`
  - A `while` loop is initiated using the `while` keyword
- `boolean expression`
  - An expression that produces a `TRUE` or `FALSE`
- `while` loop body
  - Code in the `while` loop body is only performed if the `boolean expression` is `TRUE`

```
# make sure we always sample the same random number
set.seed(15)

# initialize x such that the statement is true
x <- 0
while (x < 50) {
```

```
while (x < 50) {

  # randomly sample a number from 1 to 20
  x <- sample(1:100, 1)
  print(x)

}
```

```
[1] 37
[1] 34
[1] 38
[1] 49
[1] 5
[1] 89
```

- What do you think happens in the following?

```
x <- 0
while (TRUE) {
  x <- x + 1
}
```

`repeat` loops

`repeat` loop syntax

- The `repeat` loop...well...repeats a task forever!
- Notice that there is not condition statement. A `break` statement is used to "break" out of the loop one a condition is met

- `repeat` loop syntax

```
repeat {
repeat loop body
conditional break statement
}
```

- `repeat`
  - A `repeat` loop is initiated using the `while` keyword
- `repeat loop body`
  - Code that is executed at each iteration of the loop
- `conditional break statement`
  - A conditional statement, typically an `if` statement, that indicates whether or not we should `break` out of the loop
  - Note `break` is a keyword that tells `R` to end the loop

```
set.seed(15)

# initialize x such that the statement is true
repeat {

  # randomly sample a number from 1 to 20
  x <- sample(1:100, 1)
  print(x)

  if (x >= 50) { break }

}
```

```
[1] 37
[1] 34
[1] 38
[1] 49
[1] 5
[1] 89
```

- What do you think happens in the following?

```
x <- 0
repeat {
  x <- x + 1
}
```