loops.Rmd

Damian Sanchez

4/6/2023

For loops are another way we can tell the computer to repeat tasks for us. They are very versatile and very explicit, so that means that we are controlling everything that is run on each iteration of the loop (mostly). As opposed to apply functions where iterations happen kind of under the hood, and the apply functions can only be used to loop over (iterate) over one function. Loops can let us iterate over multiple functions and whole blocks of code.

The general structure of the for loop are

```
for (values in object_with_values) {
  do something with(value)
}
```

```
lengths <- c(13.3, 15, 100)

for (length in lengths) {
  mass <- 0.73* length^2
  print(mass)
  # we can't use return() in for loops
}
```

```
## [1] 129.1297
## [1] 164.25
## [1] 7300
```

Exercise 1 and 2

```
numbers <- c(1, 2, 3, 4, 5)
for (number in numbers){
  print(number*3)
}
```

```
## [1] 3
## [1] 6
## [1] 9
## [1] 12
## [1] 15
```

```
mass_lbs <- c(2.2, 3.5, 9.6, 1.2)
for (mass_lbs in mass_lbs) {
  mass_kg = 2.2 * mass_lbs
  print(mass_kg)
}
```

```
## [1] 4.84
## [1] 7.7
## [1] 21.12
## [1] 2.64
```

```
## Looping over an index
## What is an Index in R?
## It is the numeric position of values inside any data structure in R. For example, the following vect
flowers <- c("lilacs", "daises", "jasmins")
str(flowers)
```

```
##  chr [1:3] "lilacs" "daises" "jasmins"
```

```
## To access the second element in the vector, I need to use the index number 2 inside the square brack
flowers[2]
```

```
## [1] "daises"
```

We can use numbers as indices to loop over values inside a vector.

```
for (i in 1:3) {
  print(i)
  print(flowers[i])
}
```

```
## [1] 1
## [1] "lilacs"
## [1] 2
## [1] "daises"
## [1] 3
## [1] "jasmins"
```

Exercise #3

```
birds = c('robin', 'woodpecker', 'blue jay', 'sparrow')
for (i in 1:length(birds)){
  print(birds[i])
}
```

```
## [1] "robin"
## [1] "woodpecker"
## [1] "blue jay"
## [1] "sparrow"
```

Store the results of running a for loop using indices

So far we have just printed some values and results from some equations.

Usually what we need is to save the results of running a for loop, so that we can use them for later.

When we are using a function, what do we use to store the results of the function? We assign the result to a variable name:

```
my_result <- 0.73*lengths^2

# but for in loops we do not have that option:

# we can't do
```

but for in loops we do not have that option: we can't do

The only way to save results from each iterations of the loop is by saving them into any empty object.

```
my_results <- vector(mode = "character", length = length(flowers))
my_results
```

```
## [1] "" "" ""
```

Now we can use this empty vector and indices inside and aloop to store results

```
for(i in 1:length(flowers)) {
  upper <- toupper(flowers[i])
  my_results[i] <- upper

}
my_results
```

```
## [1] "LILACS"  "DAISES"  "JASMINS"
```

Exercise 4

```
radius <- c(1.3, 2.1, 3.5)
areas <- vector(mode = "numeric", length = length(radius))
for (i in 1:length(radius)){
  areas[i] <- pi * radius[i] ^ 2
}
areas
```

```
## [1]  5.309292 13.854424 38.484510
```

Looping over mutliple objects with indices

We have 3 vectors:

```r
dino_names <- c("T-rex", "Ankylosaur", "Triceratops")
# we have different a values for each of these dino species
a_values <- c(0.73, 5.4, 100)
b_values <- c(2, 0.5, 1.2)
dino_lengths <- c(15, 3, 20)
dino_masses <- vector(mode = "numeric", length = length(dino_names))
```

We can iterate through these values within a loop

```r
for (i in 1:length(dino_names)) {
  print(dino_names[i])
  mass <- a_values[i]* dino_lengths[i]^b_values[i]
  dino_masses[i] <- mass
  print(mass)
}
```

```
## [1] "T-rex"
## [1] 164.25
## [1] "Ankylosaur"
## [1] 9.353074
## [1] "Triceratops"
## [1] 3641.128
```

```r
dino_masses
```

```
## [1]  164.250000    9.353074 3641.128406
```

Exercise 5

```r
lengths = c(1.1, 2.2, 1.6)
widths = c(3.5, 2.4, 2.8)
areas <- vector(mode = "numeric", length = length(lengths))
for (i in 1:length(lengths)) {
  areas[i] <- lengths[i] * widths[i]
}
print(areas)
```

```
## [1] 3.85 5.28 4.48
```