CT5102: Programming for Data Analytics

2. Subsetting Vectors

Prof. Jim Duggan,
School of Computer Science
University of Galway.

https://github.com/JimDuggan/explore_or

R's subsetting operators are fast and powerful. Mastering them allows you to succinctly perform complex operations in a way that few other languages can match.

— Hadley Wickham (Wickham, 2019)

Overview

- Subsetting operators allow you to process data stored in atomic vectors and lists and R provides a range of flexible approaches that can be used to subset data.
- This lecture presents important subsetting operations, and knowledge of these is key in terms of understanding later lectures on processing lists in R

(2.1) Subsetting Atomic Vectors

First, we generate some random data (restaurant arrivals – Poisson)

```
# set the seed
set.seed(111)
# Generate the count data, assume a Poisson distribution
customers <- rpois(n = 10, lambda = 100)
# Name each successive element to be the day number
names(customers) <- paste0("D",1:10)
customers
```

```
#>  D1  D2  D3  D4  D5  D6  D7  D8  D9 D10
#> 102  96  97  98 101  85  98 118 102  94
```

(a) Using positive integers

- Positive integers will subset atomic vector elements at given locations, and this type of index can have one or more values.
- To extract the n^{th} item from a vector x the term x[n] is used
- This can also apply to a sequence, for example, a sequence of values, starting at n and finishing at m can be extracted from the vector x as x[n:m].
- Indices can also be generated using the combine function c(), which is then used to subset a vector.

```
customers
#>  D1  D2  D3  D4  D5  D6  D7  D8  D9 D10
#> 102  96  97  98 101  85  98 118 102  94
```

```
# Get the customer from day 1
customers[1]
#>  D1
#> 102

# Get the customers from day 1 through day 5
customers[1:5]
#>  D1  D2  D3  D4  D5
#> 102  96  97  98 101

# Use c() to get the customers from day 1 and the final day
customers[c(1,length(customers))]
#>  D1 D10
#> 102  94
```

(b) Using negative integers

Negative integers can be used to exclude elements from a vector, and one or more elements can be excluded

```
# Exclude the first day's observation
customers[-1]
#>  D2  D3  D4  D5  D6  D7  D8  D9 D10
#>  96  97  98 101  85  98 118 102  94

# Exclude the first and last day
customers[-c(1,length(customers))]
#>  D2  D3  D4  D5  D6  D7  D8  D9
#>  96  97  98 101  85  98 118 102
```

(c) Using logical vectors

- Logical vectors can be used to subset a vector, and this is a powerful feature of R
- This allows for the use of relational and logical operators to perform subsetting.
- The idea is a simple one: when a logical vector is used to subset a vector, only the corresponding cells of the target vector that are TRUE in the logical vector will be retained.

For example, the code below will find any value that is greater than 100.

```
# Create a logical vector based on a relation expression
lv <- customers > 100
lv
#>    D1    D2    D3    D4    D5    D6    D7    D8    D9   D10
#>  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE  TRUE FALSE

# Filter the original vector based on the logical vector
customers[lv]
#>  D1  D5  D8  D9
#> 102 101 118 102
```

Typically, these two statements are combined into the one expression, so you will often see the following style of code in R.

```
# Subset the vector to only show values great than 100
customers[customers > 100]
#>  D1  D5  D8  D9
#> 102 101 118 102
```

Recycling elements

- A convenient feature of subsetting with logical vectors is that the logical vector size does not have to equal the size of the target vector

```
# Subset every second element from the vector
customers[c(TRUE,FALSE)]
#>  D1  D3  D5  D7  D9
#> 102  97 101  98 102
```

(d) Using named elements

- If a vector has named elements, often set via the function names(), then elements can subsetted using their names.
- This is convenient if you need to retrieve an element but do not know its exact indexed location.

```
customers
#>  D1  D2  D3  D4  D5  D6  D7  D8  D9 D10
#> 102  96  97  98 101  85  98 118 102  94
# Show the value from day 10
customers["D10"]
#> D10
#>  94
```

which() function

```
# Use which() to find the indices of the true elements in the
# logical vector
v <- which(customers > 100)
v
#> D1 D5 D8 D9
#>  1  5  8  9

# Filter customers based on these indices
customers[v]
#>  D1  D5  D8  D9
#> 102 101 118 102
```

(2.2) Subsetting lists

- Subsetting lists is more challenging than subsetting atomic vectors.
- There are three methods that can be used, and to illustrate the core idea (before moving on to a more detailed example), we define a list l1 that contains three elements

```
# Create a simple vector
l1 <- list(a="Hello",
           b=1:5,
           c=list(d=c(T,T,F),
                  e="Hello World"))
# Show the structure
str(l1)
#> List of 3
#>  $ a: chr "Hello"
#>  $ b: int [1:5] 1 2 3 4 5
#>  $ c:List of 2
#>   ..$ d: logi [1:3] TRUE TRUE FALSE
#>   ..$ e: chr "Hello World"
```

(a) The [operator

- When applied to a list, the [operator will always return a list.
- The same indexing method used for atomic vectors can also be used for filtering lists, namely:
  - positive integers,
  - negative integers,
  - logical vectors, and
  - the element name.

```r
# extract the first and third element of the list l1
str(l1[c(1,3)])
#> List of 2
#>  $ a: chr "Hello"
#>  $ c:List of 2
#>   ..$ d: logi [1:3] TRUE TRUE FALSE
#>   ..$ e: chr "Hello World"
```

```r
# Show the first and third element using a logical vector
str(l1[c(TRUE,FALSE,TRUE)])
#> List of 2
#>  $ a: chr "Hello"
#>  $ c:List of 2
#>   ..$ d: logi [1:3] TRUE TRUE FALSE
#>   ..$ e: chr "Hello World"
```

```r
# exclude the first and third element of the list
str(l1[-c(1,3)])
#> List of 1
#>  $ b: int [1:5] 1 2 3 4 5
```

```r
# Show the first element using a character vector
str(l1["a"])
#> List of 1
#>  $ a: chr "Hello"
```

(b) The [[operator

- Often, we need to access the data in a list
- the [[operator extracts the contents of a list at a given location (i.e., element 1, 2, .., N), where N is the list length.

```
# extract the contents of the first list element
l1[[1]]
#> [1] "Hello"

# extract the contents of the second  list element
l1[[2]]
#> [1] 1 2 3 4 5
```

```r
# extract the contents of the third list element (a list!)
str(l1[[3]])
#> List of 2
#>  $ d: logi [1:3] TRUE TRUE FALSE
#>  $ e: chr "Hello World"

# extract the contents of the first element of the third element
l1[[3]][[1]]
#> [1]  TRUE  TRUE FALSE


# extract the contents of the second element of the third element
l1[[3]][[2]]
#> [1] "Hello World"
```

List element names can be used to subset

```
# extract the contents of the first list element
l1[["a"]]
#> [1] "Hello"

# extract the contents of the second  list element
l1[["b"]]
#> [1] 1 2 3 4 5

# extract the contents of the third list element (a list!)
str(l1[["c"]])
#> List of 2
#>  $ d: logi [1:3] TRUE TRUE FALSE
#>  $ e: chr "Hello World"
```

(c) The $ operator

- There is a convenient alternative to the [[operator, and this is the tag operator $ which can be used once a list element is named.
- For example, for our list l1 the terms l1[[1]], l1[["a"] and l1$a return the same result.
- In the general case my_list[["y]] is equivalent to my_list$y.

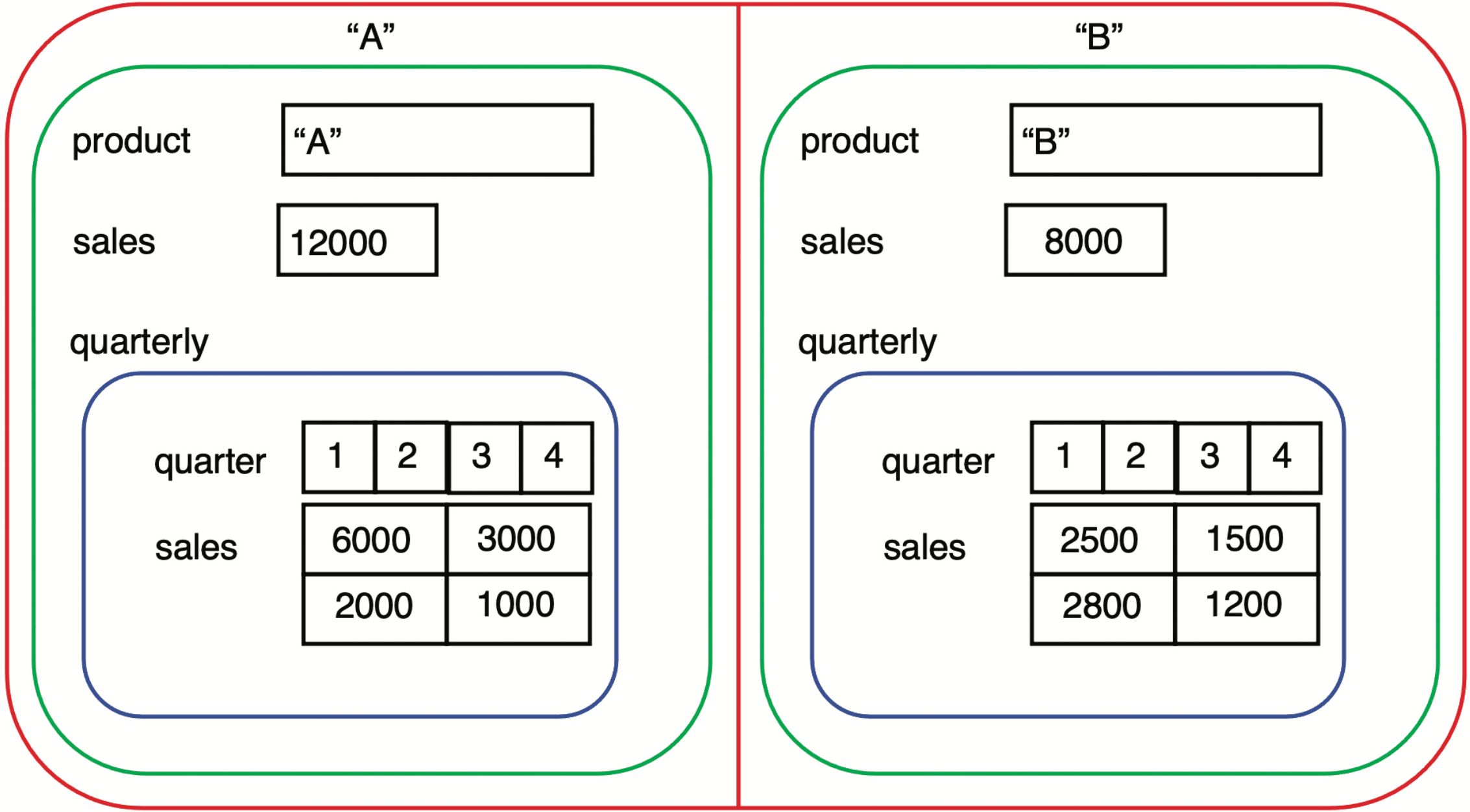
```r
# extract the contents of the first list element
l1$a
#> [1] "Hello"

# extract the contents of the second  list element
l1$b
#> [1] 1 2 3 4 5

# extract the contents of the third list element (a list!)
str(l1$c)
#> List of 2
#>  $ d: logi [1:3] TRUE TRUE FALSE
#>  $ e: chr "Hello World"

# extract the contents of the first element of the third element
l1$c$d
#> [1]  TRUE  TRUE FALSE
```

```r
# A small products database. Main list has two products
products <- list(A=list(product="A",
                        sales=12000,
                        quarterly=list(quarter=1:4,
                                       sales=c(6000,3000,2000,1000))),
                 B=list(product="B",
                        sales=8000,
                        quarterly=list(quarter=1:4,
                                       sales=c(2500,1500,2800,1200))))
str(products)
#> List of 2
#>  $ A:List of 3
#>   ..$ product  : chr "A"
#>   ..$ sales    : num 12000
#>   ..$ quarterly:List of 2
#>   .. ..$ quarter: int [1:4] 1 2 3 4
#>   .. ..$ sales  : num [1:4] 6000 3000 2000 1000
#>  $ B:List of 3
#>   ..$ product  : chr "B"
#>   ..$ sales    : num 8000
#>   ..$ quarterly:List of 2
#>   .. ..$ quarter: int [1:4] 1 2 3 4
#>   .. ..$ sales  : num [1:4] 2500 1500 2800 1200
```

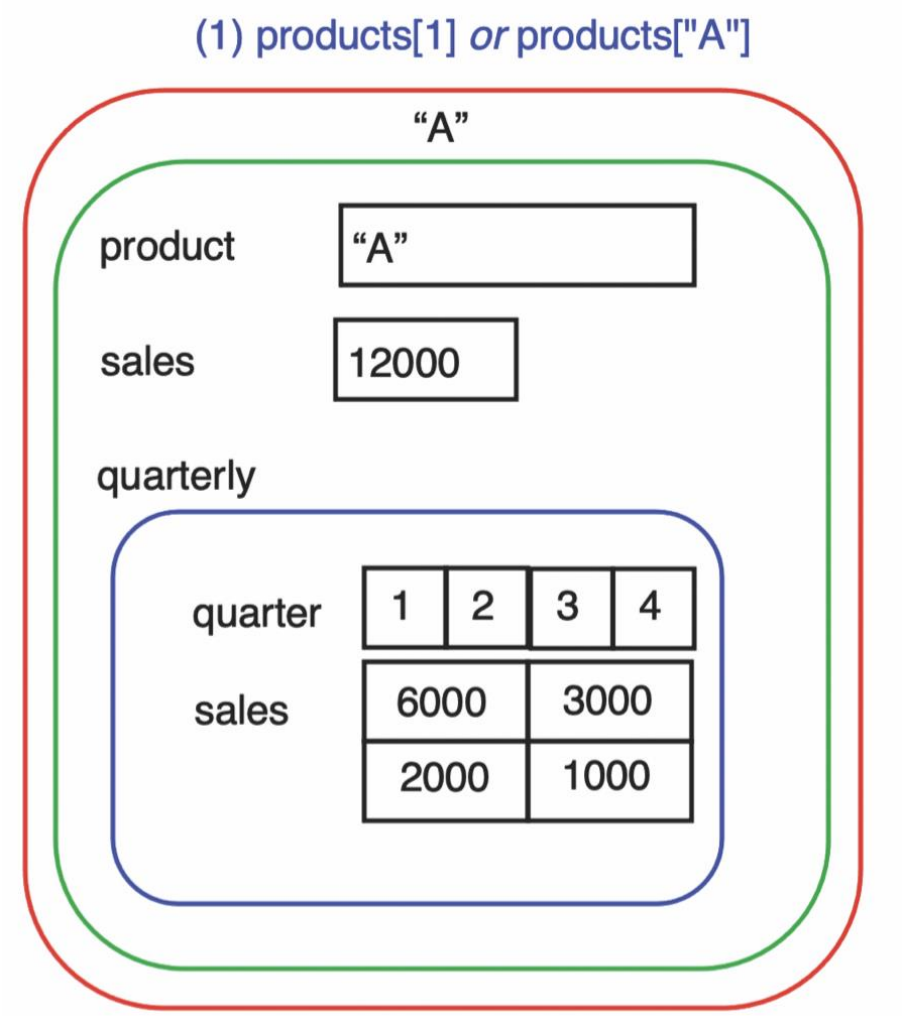

"A"
product
"A"
sales
12000
quarterly
quarter
1
2
3
4
sales
6000
3000
2000
1000
"B"
product
"B"
sales
8000
quarterly
quarter
1
2
3
4
sales
2500
1500
2800
1200

Key points

- Just a vector with 2 elements
- However, each element has significant internal structure
- A list:
  - product
  - sales
  - quarterly (a list)
    - quarter
    - sales

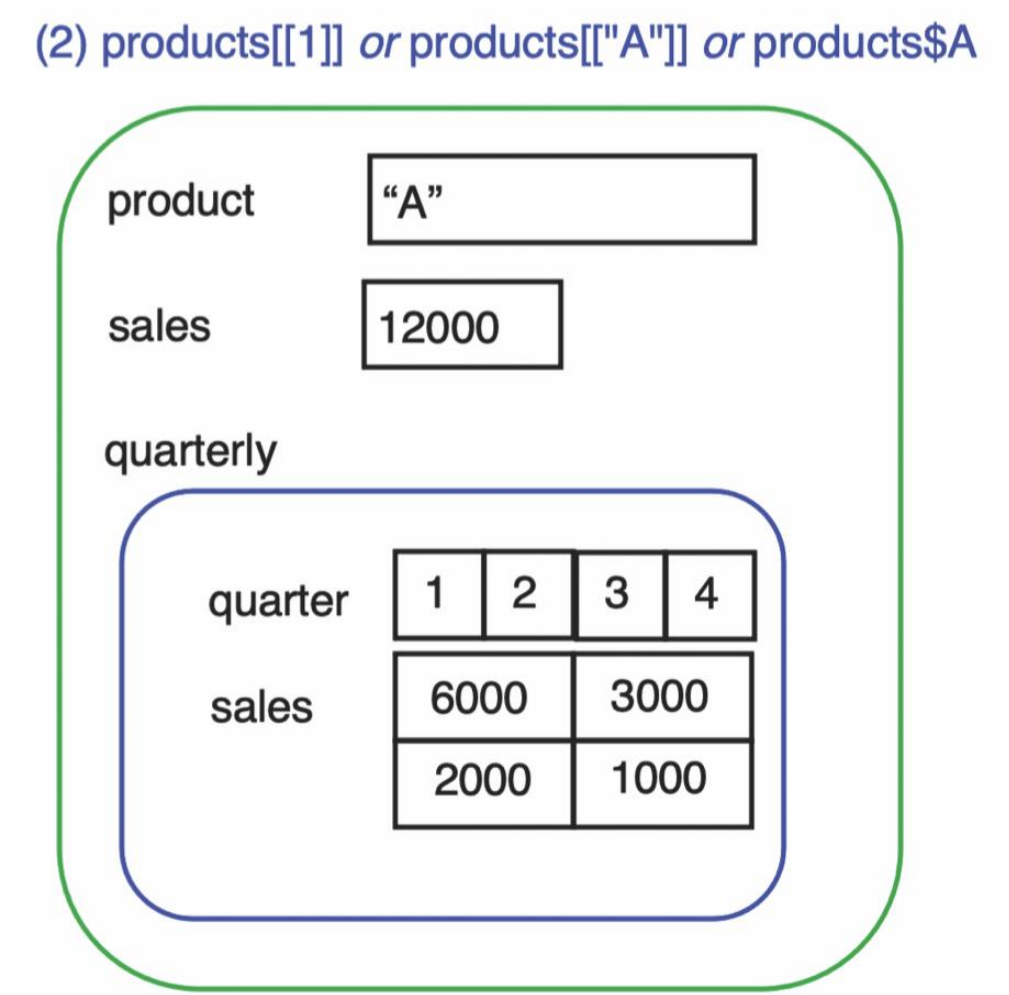
```
str(products)
#> List of 2
#>  $ A:List of 3
#>   ..$ product  : chr "A"
#>   ..$ sales    : num 12000
#>   ..$ quarterly:List of 2
#>   .. ..$ quarter: int [1:4] 1 2 3 4
#>   .. ..$ sales  : num [1:4] 6000 300
#>  $ B:List of 3
#>   ..$ product  : chr "B"
#>   ..$ sales    : num 8000
#>   ..$ quarterly:List of 2
#>   .. ..$ quarter: int [1:4] 1 2 3 4
#>   .. ..$ sales  : num [1:4] 2500 150
```

(i) Extract first element of the list




```
ex1.1 <- products[1]
ex1.2 <- products["A"]
str(ex1.1)
#> List of 1
#>  $ A:List of 3
#>   ..$ product  : chr "A"
#>   ..$ sales    : num 12000
#>   ..$ quarterly:List of 2
#>   .. ..$ quarter: int [1:4] 1 2 3 4
#>   .. ..$ sales  : num [1:4] 6000 3000 2000 1000
```

(ii) Extract the contents of the first list element




```
# Example (2) - get the contents of the first l
ex2.1 <- products[[1]]
ex2.2 <- products[["A"]]
ex2.3 <- products$A
str(ex2.1)
#> List of 3
#>  $ product  : chr "A"
#>  $ sales    : num 12000
#>  $ quarterly:List of 2
#>   ..$ quarter: int [1:4] 1 2 3 4
#>   ..$ sales  : num [1:4] 6000 3000 2000 1000
```

(iii) Extract the product name from the first list element

(3) products[[1]][[1]] *or* products[["A"]][["product"]] *or* products$A$product

product	"A"

```
# Example (3) - get the product name for t
ex3.1 <- products[[1]][[1]]
ex3.2 <- products[["A"]][["product"]]
ex3.3 <- products$A$product
str(ex3.1)
#>  chr "A"
```

(iv) Extract the annual sales of product A

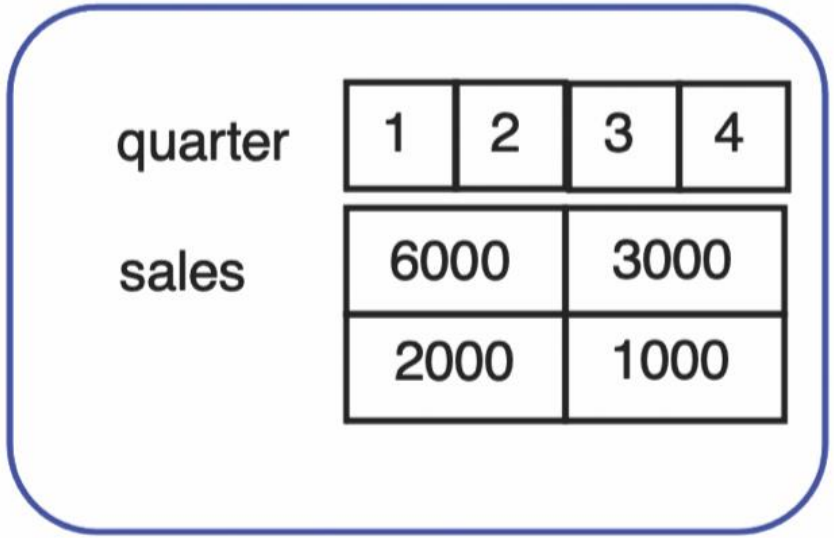
(4) products[[1]][[2]] *or* products[["A"]][["sales"]] *or* products$A$sales

sales	12000

```
# Example (4) - get the annual sales for
ex4.1 <- products[[1]][[2]]
ex4.2 <- products[["A"]][["sales"]]
ex4.3 <- products$A$sales
str(ex4.1)
#>  num 12000
```

(v) Extract the third element of the first element

(5) products[[1]][[3]] *or* products[["A"]][["quarterly"]] *or* products$A$quarterly

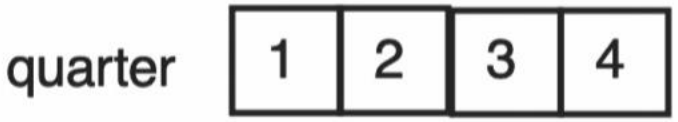


quarter	1	2	3	4
sales	6000		3000	
	2000		1000	

```
# Example (5) - get as a list, the detailed qu
ex5.1 <- products[[1]][[3]]
ex5.2 <- products[["A"]][["quarterly"]]
ex5.3 <- products$A$quarterly
str(ex5.1)
#> List of 2
#>  $ quarter: int [1:4] 1 2 3 4
#>  $ sales  : num [1:4] 6000 3000 2000 1000
```

(vi) access the inner list values, again by using [[or $

(6) products[[1]][[3]][[1]] *or*
products[["A"]][["quarterly"]][["quarter"]] *or*
products$A$quarterly$quarter




```
# Example (6) - get the quarters
ex6.1 <- products[[1]][[3]][[1]]
ex6.1 <- products[["A"]][["quarterly"]][["quarter"]]
ex6.1 <- products$A$quarterly$quarter
str(ex6.1)
#>  int [1:4] 1 2 3 4
```

(vii) Access the quarterly sales

(7) products[[1]][[3]][[2]] or products[["A"]][["quarterly"]][["sales"]] *or* products$A$quarterly$sales

sales	6000	3000
	2000	1000

```
# Example (7) - get the quarterly sales
ex7.1 <- products[[1]][[3]][[2]]
ex7.1 <- products[["A"]][["quarterly"]][["sales"]]
ex7.1 <- products$A$quarterly$sales
str(ex7.1)
#>  num [1:4] 6000 3000 2000 1000
```

(viii) subset the first two quarterly sales values

(8) products[[1]][[3]][[2]][1:2] *or*
products[["A"]][["quarterly"]][["sales"]][1:2] *or*
products$A$quarterly$sales[1:2]

sales	6000	3000

```
# Example (8) - get the quarterly sales for the first t
ex8.1 <- products[[1]][[3]][[2]][1:2]
ex8.2 <-products[["A"]][["quarterly"]][["sales"]][1:2]
ex8.3 <-products$A$quarterly$sales[1:2]
str(ex8.1)
#>  num [1:2] 6000 3000
```

(2.3) Iteration with loop, and if statement

- There are a number of basic looping structures than can be used in R, and we will focus on one of these, the for loop.
- The general structure is for (var in seq) expr, where:
  - var is a name for a variable that will change its value for each loop iteration.
  - seq is an expression that evaluates to a vector – e.g. seq_along()
  - expr is an expression, which can be either a simple expression, or a compound expression of the form {expr1; expr2}, which is effectively a number of lines of code with two curly braces.

seq_along() – can generate indices

```
set.seed(100)
v <- sample(1:6,10,replace = T)
v
#>  [1] 2 6 3 1 2 6 4 6 6 4

sa <- seq_along(v)
sa
#>  [1]  1  2  3  4  5  6  7  8  9 10
```

for loop example

```r
v <- sample(1:6,10,replace = T)
v
#>  [1] 2 6 3 1 2 6 4 6 6 4
```

```r
n_six <- 0
for(i in seq_along(v)){
  n_six <- n_six + as.integer(v[i] == 6)
}
n_six
#> [1] 4
```

if statement

- When iterating through individual vector elements, you may need to execute a statement based on a current vector value.
- To do this, the if statement can be used, and there are two main forms:
  - if (cond) expr which evaluates expr if the condition cond is tru
  - if (cond) true.expr else false.expr, which evaluates true.expr if the condition is true, and otherwise evaluates false.expr

Loop and if statement example…

```
# create a test vector
v <- 1:10
# create a logical vector to store the result
lv <- vector(mode="logical",length(v))
# Loop through the vector, examining each element
for(i in seq_along(v)){
  if(v[i] > mean(v))
    lv[i] <- TRUE
  else
    lv[i] <- FALSE
}
v[lv]
#> [1]  6  7  8  9 10
```

(2.4) Mini-case: Star Wars movies

- The aim of this mini-case is to show how we can subset lists
- The CRAN package repurrrsive is used, which contains Star Wars related information
  - sw_films
  - sw_people
  - sw_planets
  - sw_species
  - sw_starships

sw_films – A list (7), each with a list [14]

```
library(repurrrsive)
length(sw_films)
#> [1] 7
# show the list elements for 1st element
names(sw_films[[1]])
#>  [1] "title"          "episode_id"    "opening_crawl" "director"
#>  [5] "producer"       "release_date"  "characters"    "planets"
#>  [9] "starships"      "vehicles"      "species"       "created"
#> [13] "edited"         "url"
```

```
# Get the first film name and movie director
sw_films[[1]]$title
#> [1] "A New Hope"
sw_films[[1]]$director
#> [1] "George Lucas"
```

Aim – find George Lucas movies

- A for-loop structure is used to iterate over the entire loop and mark those 7 elements as either a match (TRUE) or not a match (FALSE). This information is stored in an atomic vector.
- Before entering the loop, we create a logical vector variable (is_target) of size 7 (the same size as the list), and this will store information on whether a list item should be marked for further processing.
- For each list element we extract the director's name, check if it matches the target ("George Lucas"), and store this logical value in the corresponding element of is_target.
- The vector is_target can then be used to filter the original sw_films list and retain all the movies directed by George Lucas.

Code solution

```
# Search for movies by George Lucas and store these in a new list
target <- "George Lucas"
# Create a logical vector to hold information for positive matches
is_target <- vector(mode="logical",length = length(sw_films))
# Iterate through the entire sw_films list (of 7)
for(i in seq_along(sw_films)){
  is_target[i] <- sw_films[[i]]$director == target
}
target_list <- sw_films[is_target]
```

```r
# Create a movies vector to store the movie names
movies <- vector(mode="character",length = length(target_list))
# Iterate through the list to extract the movie title
for(i in seq_along(target_list)){
  movies[i]<-target_list[[i]]$title
}
movies
#> [1] "A New Hope"           "Attack of the Clones"
#> [3] "The Phantom Menace"   "Revenge of the Sith"
```

Another solution...

```
# Create a new list to store the data in a different way
sw_films1 <- list(title=c(),
                  episode_id=c(),
                  director=c())
# Iterate through the list to append the title and director
for(i in seq_along(sw_films)){
  sw_films1$title      <- c(sw_films1$title,
                            sw_films[[i]]$title)
  sw_films1$episode_id <- c(sw_films1$episode_id,
                            sw_films[[i]]$episode_id)
  sw_films1$director   <- c(sw_films1$director,
                            sw_films[[i]]$director)
}
```

```
sw_films1
#> $title
#> [1] "A New Hope"                "Attack of the Clones"
#> [3] "The Phantom Menace"        "Revenge of the Sith"
#> [5] "Return of the Jedi"        "The Empire Strikes Back"
#> [7] "The Force Awakens"
#>
#> $episode_id
#> [1] 4 2 1 3 6 5 7
#>
#> $director
#> [1] "George Lucas"     "George Lucas"     "George Lucas"
#> [4] "George Lucas"     "Richard Marquand" "Irvin Kershner"
#> [7] "J. J. Abrams"
```

Find George Lucas Movies

```
# Find all the movie titles by George Lucas
sw_films1$title[sw_films1$director=="George Lucas"]
#> [1] "A New Hope"              "Attack of the Clones"
#> [3] "The Phantom Menace"      "Revenge of the Sith"
```

(2.5) Summary Functions

Function	Description
as.list()	Coerces the input argument into a list.
paste0()	Converts and concatenates arguments to character strings.
rpois()	Generates Poisson random numbers (mean lambda).
seq_along()	Generates a sequence to iterate over vectors.
which()	Provide the TRUE indices of a logical object.

(2.6) Exercises

2. Filter the list `sw_people` (87 elements), contained in `repurrrsive` to include only those whose height is *not unknown*. Use an atomic vector `has_height` to filter the list, and populate this vector using a loop structure. This new list (`sw_people1`) should have 81 elements.

```
sum(has_height)
#> [1] 81
length(sw_people1)
#> [1] 81
```

3. Using a `for` loop over the filtered list `sw_people1` from exercise 2, create a list of people whose height is greater than or equal to 225 inches. The resulting vector should grow as matches are found, as we do not know in advance how many people will be contained in the result. Use the command `characters <- c()` to create the initial empty result vector.

The following result should be obtained.

```
# These are the characters whose height is >= 225
characters
#> [1] "Chewbacca"   "Yarael Poof" "Lama Su"     "Tarfful"
```

4. Using a `for` loop to iterate over the list `sw_planets` and display those planets (in a character vector) whose diameter is greater than or equal to the mean. Use a pre-processing step that will add a new list element to each planet, called `diameter_numeric`. This pre-processing step can also be used to calculate the mean diameter. Also, use the pre-processing step to keep track of those planets whose diameter is "unknown", and use this information to create an updated list `sw_planets1` that excludes all the values.

You can check your solution against the following values.

```
# The list elements that will be excluded (diameter unknown)
exclude
#>  [1] 37 39 42 44 45 46 47 49 50 51 52 53 54 55 57 59 61
# The mean diameter
mean_diameter
#> [1] 8936
# The first three and last three planets returned
gte_mean[c(1:3,(length(gte_mean)-2):(length(gte_mean)))]
#> [1] "Alderaan"   "Yavin IV"   "Bespin"     "Muunilinst" "Kalee"
#> [6] "Tatooine"
```

5. Based on the list `sw_species`, and given that each species has a `classification`, create the following tabular summary, again using a loop to iterate through the list. Make use of the `table()` function that was covered in Chapter 2 to present the summary.

```
# A tabular summary of the types of species
t_species
#> c_species
#>  amphibian artificial  gastropod  insectoid     mammal    mammals
#>          6          1          1          1         16          1
#>    reptile  reptilian   sentient    unknown
#>          3          1          1          6
```