Lecture 2: An overview of `R`: part I

This lecture gives an overview of `R` and introduces some basic characteristics of `R`. This includes

1. Basic computations in `R`
2. How to create an object
3. Data types
4. How to generate data
5. Operators
6. Learn packages for EPIB 607

2.1 Basic computations in `R`

In [1]:

```
2+3
```

5

In [2]:

```
2*3
```

6

In [3]:

```
log(4)     # Natural log
```

1.38629436111989

In [4]:

```
exp(2)
```

7.38905609893065

In [5]:

```
2e3
```

2000

In [6]:

```
2^3; 8^(1/3)
```

8

2

In [7]:

```
sqrt(4)
```

2

2.2 Create an R object

We cannot always work with numbers by copy-paste. Create R objects to store the numbers -> data.

Not only numbers but also text, date, etc. are data that R can use.

In [8]:

```
x <- 5; x
```

5

In [9]:

```
y <- z <- 6
y
z
```

6

6

In [10]:

```
peak.no <- 21
course = "EPIB 613"
"Yi" -> me            #  <- , = and -> are equivalent when we assign values
cat(c(me, "teaches", peak.no, "students in", course))
# Don't worry about cat(). If you do, run ?cat in R.
```

```
Yi teaches 21 students in EPIB 613
```

Calculations with stored R objects

Example: calculate the number of students left in Yi's class.

In [11]:

```
peak.no <- no.students <- 21
no.quit <- 3
no.stay <- peak.no - no.quit
no.stay
```

18

Advantages:

- Most importantly, re-use the values by simply calling the the object. Reproducibility!
- Can use variable names that make sense to yourself - Very clear and can be easily edited later.

Note:

- Whether or not to store data in a named `R` object is totally up to you.

R is case sensitive.

In [12]:

```
Course <- "EPIB 601"
course
```

'EPIB 613'

The old value will be replaced by the new one.

In [13]:

```
print(no.students)
```

```
[1] 21
```

In [14]:

```
no.students <- no.stay
print(no.students)
```

```
[1] 18
```

Rule for creating an object:

- Variables can be alphabetic or alphanumeric, but not numeric (you are not allowed to create numeric variables).
- There are no restrictions to the length of the variable name.
- Do NOT assign the single letter names c, g, t, C, D, F, I and T as they are default names that are used by R. For instance, T and F are abbreviations for TRUE and FALSE in logical operations. We should avoid using names that are already used by the system.

2.3 Data types and structures

2.3.0 Data types

In [15]:

```
number <- c(1, 2, 3)
class(number)
```

'numeric'

In [16]:

```
# As in most programming languages, there are integers and floating-point number
s in R
class(5L)
```

'integer'

In [17]:

```
# Double precision floating-point numbers in R
# is.double() checks whether an object is a double precision floating-point numb
er
is.double(5); is.double(5L)
```

TRUE

FALSE

In [18]:

```
# How precise is double precision?
options(digits = 22) # show more decimal points
print(1/3)
options(digits = 7) # reset to default
```

```
[1] 0.3333333333333333148296
```

In [19]:

```
letters <- letters[1:3]; print(letters)
class(letters)
```

```
[1] "a" "b" "c"
```

'character'

In [20]:

```
logical <- c(TRUE, FALSE)
class(logical)
```

'logical'

In [21]:

```
factor <- as.factor(letters[1:3]); print(factor)
class(factor)
```

```
[1] a b c
Levels: a b c
```

'factor'

2.3.1* Scalar

Not considered as a stand-alone data structure because it is basically a vector of length 1.

In [22]:

```
x <- 5; x
```

5

2.3.2 Vector

In R, we work with vectors.

In [23]:

```
# As a big fan of winter sports, I hope that...
snow.days.per.week.mtl <- c(7, 7, 7, 7)
print(snow.days.per.week.mtl)
```

```
[1] 7 7 7 7
```

In [24]:

```
# We can add names to the vector for each entry
names(snow.days.per.week.mtl) <- rep("Jan 2018", 4)
print(snow.days.per.week.mtl)
```

```
Jan 2018 Jan 2018 Jan 2018 Jan 2018 
       7        7        7        7 
```

In [25]:

```
# But the names will not affect calculations.
sum(snow.days.per.week.mtl)
```

28

2.3.3 Matrix

In [26]:

```
mymatrix1 <- matrix(c(3:14), nrow = 4, byrow = TRUE)
print(mymatrix1)
```

```
     [,1] [,2] [,3]
[1,]    3    4    5
[2,]    6    7    8
[3,]    9   10   11
[4,]   12   13   14
```

In [27]:

```
mymatrix2 <- matrix(c(3:14), nrow = 4, byrow = FALSE)
print(mymatrix2)
```

```
     [,1] [,2] [,3]
[1,]    3    7   11
[2,]    4    8   12
[3,]    5    9   13
[4,]    6   10   14
```

In [28]:

```
rownames <- c("row1", "row2", "row3", "row4")
colnames <- c("col1", "col2", "col3")
rownames(mymatrix1) <- rownames
colnames(mymatrix1) <- colnames
print(mymatrix1)
```

```
     col1 col2 col3
row1    3    4    5
row2    6    7    8
row3    9   10   11
row4   12   13   14
```

2.3.4 Array

Mathematically, scalars, vectors and matrices are all arrays of different dimensions

- Scalar: 1 x 1 array
- Vector of length k: 1 x k array
- Matrix of dimension m x n: m x n array

`R` treats every array below 3 dimensions differently but they are essentially not very different. Python treats them in the same way.

Now let's look at a 3-dimensional array.

In [29]:

```
myarray <- array(1:24, dim = c(4,3,2))
print(myarray)
```

```
, , 1

     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12

, , 2

     [,1] [,2] [,3]
[1,]   13   17   21
[2,]   14   18   22
[3,]   15   19   23
[4,]   16   20   24
```

A demonstration of high dimensional arrays - why is it useful?

Fake data:

- Disease: 1=Yes, 0=No
- Drug: 1=Exposed, 0=Unexposed
- BMI category: 1,2,3
- Age category: 1,2,3,4

In [30]:

```
# Don't worry about the data generating process.
set.seed(613) # Make random numbers generated from sample() reproducible.
# Randomly assign ~20% of patients to have disease.
disease <- sample(c(0,1), size = 100, replace = TRUE, prob = c(0.2, 0.8))
# Randomly assign ~40% of patients to take drug.
drug <- sample(c(0,1), size = 100, replace = TRUE, prob = c(0.4, 0.6))
bmi.cat <- sample(1:3, size = 100, replace = TRUE) # Randomly assign BMI categor
ies
age.cat <- sample(1:4, size = 100, replace = TRUE) # Randomly assign age categor
ies
data <- data.frame(drug, disease, bmi.cat, age.cat) # Make our data frame
head(data)

# The table below shows the first 6 rows of the fake dataset.
# This is a typical dataset you will see in Epidemiology.
# Each row is a patient, with their own information.
# Goal is to assess the association between disease and drug (drug safety).
```

A data.frame: 6 × 4

drug <dbl>	disease <dbl>	bmi.cat <int>	age.cat <int>
1	0	3	3
1	0	3	4
1	1	2	1
0	0	1	2
0	1	3	2
1	0	2	1

In [31]:

```r
# By tabulating the data, we can assess the association (EPIB 601 material).
# If we only tabulate drug and disease, we get a 2x2 table, which is a matrix or
a 2-dimensional array.
# 1st dimension: drug, 2nd dimension: disease
table(data[c("drug","disease")])
```

```
    disease
drug  0  1
   0 11 31
   1 19 39
```

In [32]:

```r
# This may not be enough, we want to see how people with different BMI may diffe
r (confounder, also 601 material).
# We now need a 2x2x3 table, which is a 3-dimensional array.
# 1st dimension: drug, 2nd dimension: disease, 3rd dimension: bmi.cat
table(data[c("drug", "disease", "bmi.cat")])
```

```
, , bmi.cat = 1

    disease
drug  0  1
   0  3  7
   1  5 15

, , bmi.cat = 2

    disease
drug  0  1
   0  3 12
   1  8 11

, , bmi.cat = 3

    disease
drug  0  1
   0  5 12
   1  6 13
```

In [33]:

```r
# Further include age to see how age category comes into the association
# We now need a 2x2x3x4 table, which is a 4-dimensional array.
# 1st dimension: drug, 2nd dimension: disease, 3rd dimension: bmi.cat, 4th dimen
sion: age.cat
# table(data)
```

2.3.5 Data frames

Data frame is the most commonly used member of the data types family in R. A data frame is a generalization of a matrix, in which different columns may have different modes. All elements of any column must have the same mode, i.e. all numeric or all factor, or all character.

In [34]:

```
names <- c("Lucy", "John", "Mark", "Candy")
score <- c(67, 56, 87, 91)
pass <- c(T, F, T, T)
df <- data.frame(names, score, pass); print(df)
```

```
  names score  pass
1  Lucy    67  TRUE
2  John    56 FALSE
3  Mark    87  TRUE
4 Candy    91  TRUE
```

In [35]:

```
str(df) # checking the structure of an object
```

```
'data.frame':	4 obs. of  3 variables:
 $ names: Factor w/ 4 levels "Candy","John",..: 3 2 4 1
 $ score: num  67 56 87 91
 $ pass : logi  TRUE FALSE TRUE TRUE
```

Create these data structures

```
- c
- matrix
- array
- data.frame
- ...
```

Conversion between these data structures

```
- as.vector
- as.matrix
- as.array
- as.data.frame
- ...
```

Check whether your R object has certain data structure

```
- is.vector
- is.matrix
- is.array
- is.data.frame
- ...
```

Exercise

1. Create your own vectors, matrices, arrays and data frames
2. Convert your data into different shapes
   - What will you get if you try to make your higher dimensional objects into a vector
   - How to re-format you vector into a matrix in your desired order
   - What will you get if you try to convert your data frame (with columns of different data types) into a matrix?

2.3.6 List

In above data structures, data types and dimensions have to match. But not in lists.

In [36]:

```
mylist <- list("Red", factor(c("a","b")), c(21,32,11), TRUE)
print(mylist)
```

```
[[1]]
[1] "Red"

[[2]]
[1] a b
Levels: a b

[[3]]
[1] 21 32 11

[[4]]
[1] TRUE
```

In [37]:

```
str(mylist)
```

```
List of 4
 $ : chr "Red"
 $ : Factor w/ 2 levels "a","b": 1 2
 $ : num [1:3] 21 32 11
 $ : logi TRUE
```

2.3.7* Factors

Factor is considered as a data structure - among vectors, matrices, etc.

In my opinion, factor is a data type.

Doesn't really matter.

In [38]:

```
ch.letter <- letters[1:3]
print(ch.letter)
```

```
[1] "a" "b" "c"
```

In [39]:

```
class(ch.letter)
```

'character'

In [40]:

```
fac.letter <- as.factor(letters[1:3])
print(fac.letter)
# Note the additional 'Levels: a b c' in the output
```

```
[1] a b c
Levels: a b c
```

In [41]:

```
class(fac.letter)
# Should factor be considered as a data structure or a data type?
```

'factor'

2.4 How to generate data

Combinations of the following

- c( )
- seq( )
- rep( )
- sequence( )

In [42]:

```
c(-1, 5.44, 100, 34123)
```

-1 5.44 100 34123

In [43]:

```
-1:10 # Integers, by increments of 1.
```

-1 0 1 2 3 4 5 6 7 8 9 10

In [44]:

```
seq(from = 0.33, to = 9.33, by = 3)
```

0.33 3.33 6.33 9.33

In [45]:

```
seq(from = 0, to = 1, length = 5)
```

0 0.25 0.5 0.75 1

In [46]:

```
rep(1.2, times = 5)
```

1.2 1.2 1.2 1.2 1.2

In [47]:

```
rep(c("six", "one", "three"), times = 2)
```

'six' 'one' 'three' 'six' 'one' 'three'

In [48]:

```
c(6, 1, 3, rep(seq(from = 3, to = 5, by = 0.5), times = 2))
```

6 1 3 3 3.5 4 4.5 5 3 3.5 4 4.5 5

In [49]:

```
sequence(5)
```

1 2 3 4 5

In [50]:

```
sequence(c(6, 1, 3))
```

1 2 3 4 5 6 1 1 2 3

Now we can group these vectors into higher dimensional data structures we just learned.

Exercise

1. Make a character variable of 5 student names - a, b, c, d, e
2. Make a numeric variable of their EPIB 607 scores - 80, 99, 55, 70, 84
3. Make a numeric variable of their EPIB 613 scores - 85, 90, 62, 60, 88
4. Make a factor variable with two levels - 'EPIB607' and 'EPIB613'
5. Make two numeric variable of their curved scores - $\sqrt{score} \times 10$
6. Make a logical variable of their pass/fail situation before curving
7. Make a logical variable of their pass/fail situation after curving
8. Put the four numeric vectors into a matrix, each row is a student
9. Assemble all information into a data frame, each row is a student, with columns indicating the course, score before curving, score after curving, pass/fail. Each student will have two rows.

2.5 Operators

2.5.1 Arithmetic operators

Vector operations

In [51]:

```
a <- c(1, 8, 8)
b <- c(2, 8, 4)
```

In [52]:

```
a+1 # here 1 is considered as a vector (1, 1, 1)
```

2 9 9

In [53]:

```
a+b
```

3 16 12

In [54]:

```
a*b
```

2 64 32

In [55]:

```
a^2
```

1 64 64

Operations between corresponding entries.

Matrix operations

In [56]:

```
c <- matrix(c(1,2,3,4), nrow = 2, byrow = T)
d <- matrix(c(5,6,7,8), nrow = 2, byrow = F)
print(c); print(d)
```

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

In [57]:

```
c+1
```

A
matrix:
2 × 2
of type
dbl

2	3
4	5

In [58]:

```
c+d
```

A
matrix:
2 × 2 of
type dbl

6	9
9	12

In [59]:

```
c*d
```

A matrix: 2 × 2 of type dbl

5	14
18	32

In [60]:

```
c^2
```

A matrix: 2 × 2 of type dbl

1	4
9	16

Again, operations between corresponding entries.

Exercise

Try taking the sum of a matrix and a vector, what does `R` do it?

(Optional) If you know linear algebra - cross product, dot product, matrix transpose, diagnol, determinant, rank, etc..

In [61]:

```
a %*% b
```

A matrix: 1 × 1 of type dbl

98

In [62]:

```
a %o% b
```

A matrix: 3 × 3 of type dbl

2	8	4
16	64	32
16	64	32

In [63]:

```
c %*% d
```

A matrix: 2 × 2 of type dbl

17	23
39	53

In [64]:

```
c; t(c)
```

A matrix: 2 × 2 of type dbl

1	2
3	4

A matrix: 2 × 2 of type dbl

1	3
2	4

In [65]:

```
diag(c)
```

1 4

In [66]:

```
det(c)
```

-2

2.5.2 Logical operators

In [67]:

```
# Recall vector a and b from above.
print(a); print(b)
```

```
[1] 1 8 8
[1] 2 8 4
```

In [68]:

```
a == b # Equal or not?
```

FALSE TRUE FALSE

In [69]:

```
a != b # Not equal?
```

TRUE FALSE TRUE

In [70]:

```
a > b
```

FALSE FALSE TRUE

In [71]:

```
a <= b
```

TRUE TRUE FALSE

In [72]:

```
# And
a; b
a>5 & b>5
```

1 8 8

2 8 4

FALSE TRUE FALSE

In [73]:

```
# Or
a>=5 | b>=5
```

FALSE TRUE TRUE

In [74]:

```
"ABC" == "ABC"
```

TRUE

In [75]:

```
"ABC" == "abc"
```

FALSE

In [76]:

```
TRUE + TRUE + FALSE # True = 1, False = 0.
```

2

2.6 `R` packages: `ggformula` and `mosaic`

For EPIB 607