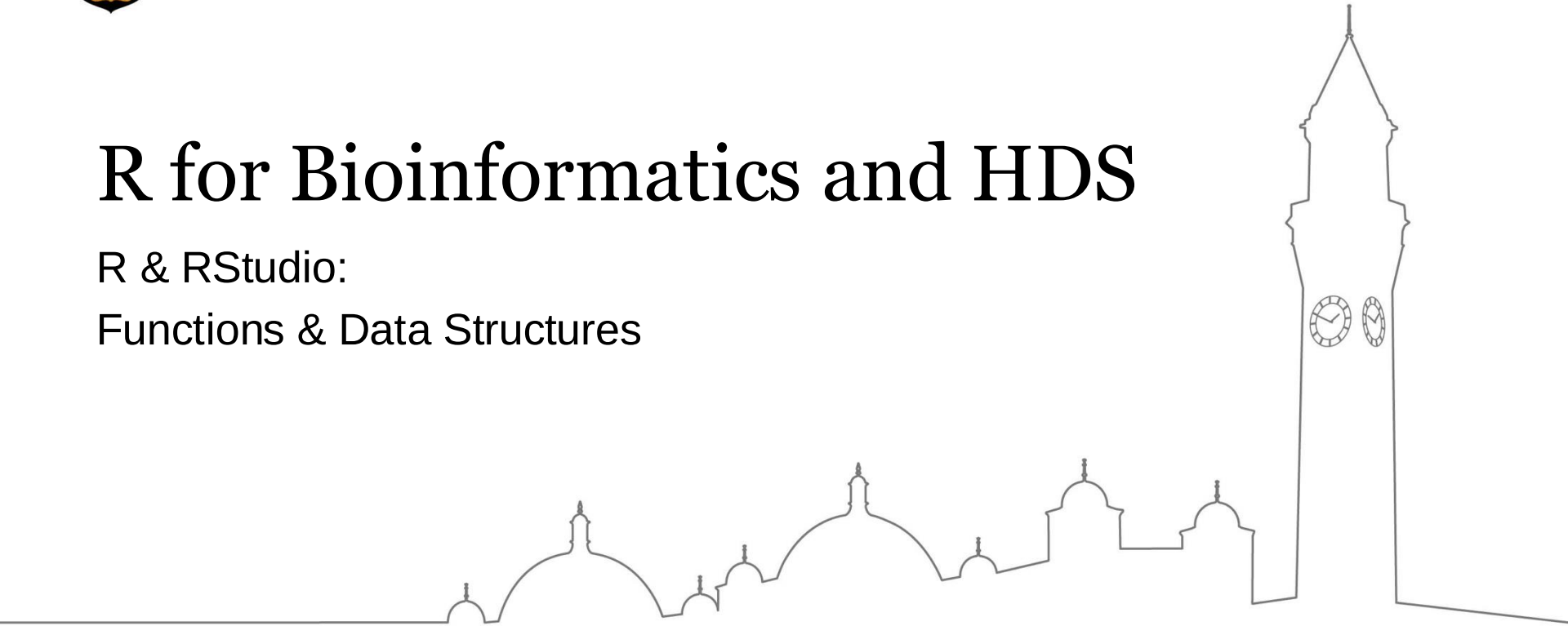
UNIVERSITY OF BIRMINGHAM

R for Bioinformatics and HDS

R & RStudio:

Functions & Data Structures



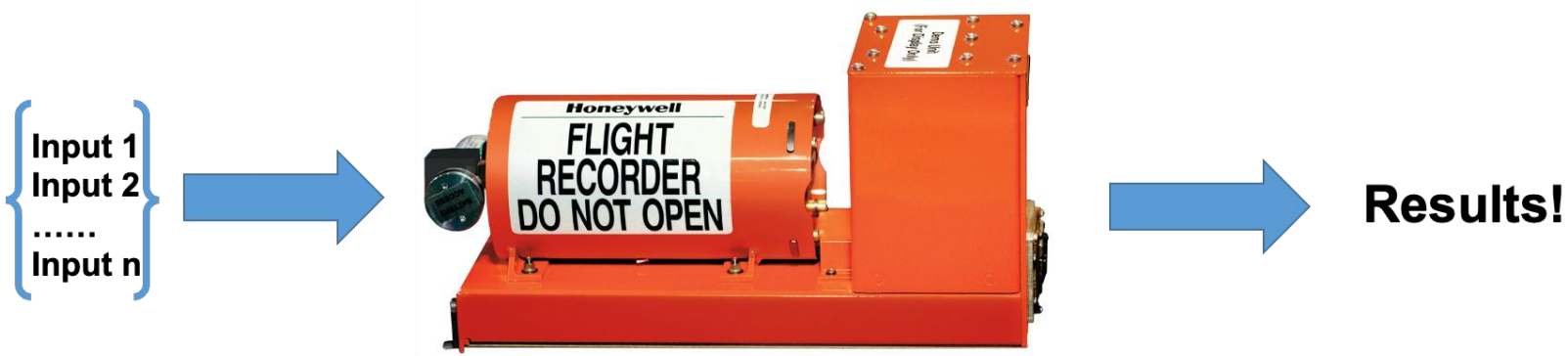
Vasileios Panagiotis Lenis
Laura Bravo Merodio

Course Structure

- Introduction to R & RStudio
- Syntax, Comments, Variables, Data Types and Operators
- Conditions, Loops, Functions and Data Structures
- Working with Data frames

What are the functions?

- Functions are “tined” scripts (black boxes) that do the “dirty work” for us.



A small taste of functions

```
sin(1)  # trigonometry functions

[1] 0.841471
```

```
log(8)  #logarithmic functions

[1] 2.079442
```

- But what if I want the log2 of a number?

```
log(8, base = 2)

[1] 3
```

UNIVERSITY OF BIRMINGHAM

Functions parameters

- A function can manipulate your output based on your needs by using different parameters.
- No need to remember these parameters:
  - Ask R for help:

```
help("log")
```

or

```
?log
```

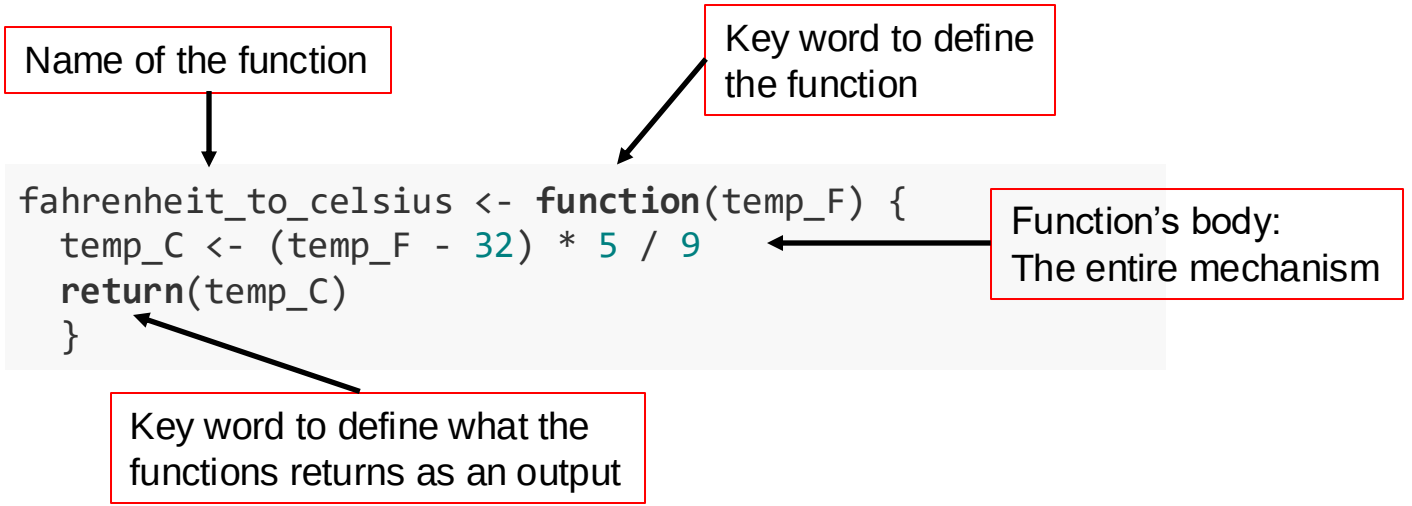
  - Use “args” function on the function you need to get the entire list of parameters

```
args(log)
```

More on built-in functions later

Creating our own functions

- You can write your own functions in order to make repetitive operations using a single command.



General structure of a function

```
func_name <- function (argument) {

  Statement 1
  Statement 2
  ....
  Statement n

  return (something)
}
```

UNIVERSITY OF BIRMINGHAM

How to use (call) them

- Let's try running our function. Calling our own function is no different from calling any other function:

```
# freezing point of water
fahrenheit_to_celsius(32)
```

```
# boiling point of water
fahrenheit_to_celsius(212)
```

UNIVERSITY OF BIRMINGHAM

Data Structures

- R is using several data structures to organize the data
- The most important structures that it uses are:
  - Vector
  - Factor
  - Matrix
  - Data frame

Vector

- The most common and basic data structure in R and is pretty much the workhorse of R.
- Two types: **Atomic Vectors** (the most common type) and **Lists**
- A collection of elements that are most commonly of mode character, logical, integer or numeric.
- Can be created with the combined function **c()** and can include the same of different type of entrances.

Vector examples

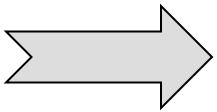
Creating a vector named “my_vector”

```
my_vector <- c(1,2,3,4)
my_vector
```

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

With the functions **typeof()**, **length()**, **class()** and **str()**
we can retrieve useful information about the size and the type of vector’s elements

```
typeof(my_vector)
str(my_vector)
length(my_vector)
class(my_vector)
```

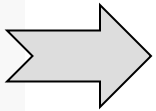


```
[1] "double"
num [1:4] 1 2 3 4
[1] 4
[1] "numeric"
```

Vector: Adding Elements

- The function **c()** (for combine) can also be used to add elements to a vector

```
z <- c("Sarah", "Tracy", "Jon")
print(z)
z <- c(z, "Annette")
print(z)
```



```
[1] "Sarah" "Tracy" "Jon"

[1] "Sarah"   "Tracy"   "Jon"     "Annette"
```

- What Happens When You Mix Types Inside a Vector?

R will create a resulting vector with a mode that can most easily accommodate all the elements it contains. This conversion between modes of storage is called “coercion”.

```
xx <- c(1.7, "a")
str(xx)
```

```
chr [1:2] "1.7" "a"
```

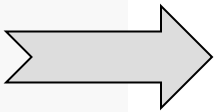
UNIVERSITY OF BIRMINGHAM

Factors

- Factors are used to represent categorical data. Factors can be ordered or unordered and are an important class for statistical analysis and for plotting.

(Factors are stored as integers, and have labels associated with these unique integers)

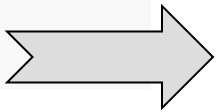
```
food <- factor(c("low", "high", "medium",
"high", "low", "medium", "high"))

levels(food)


nlevels(food)
```



```
[1] "high"   "low"    "medium”

[1] 3
```

- You can change the order that the levels appear.

```
food <- factor(food, levels = c("low",
"medium", "high"))

levels(food)
```

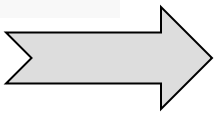


```
[1] "low"    "medium" "high"
```

Matrix

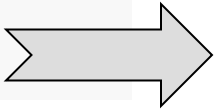
- Matrices are an extension of the numeric or character vectors. They are not a separate type of object but simply an atomic vector with dimensions; the number of rows and columns. The elements of a matrix must be of the same data type.

```
m <- matrix(1:6, nrow = 2, ncol = 3)
m
```



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

```
typeof(m)

dim(m)
```

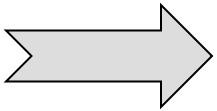


```
[1] "integer"

[1] 2 3
```

Data frames

- A data frame is a very important data type in R. It's pretty much the *de facto* data structure for most tabular data and what we use for statistics. A data frame is a collection of vectors where every vector has same length.
- We can create a data frame by hand, but usually we are using functions to import data as data frames (more about this, later).

```
df <- data.frame(id = letters[1:10], x = 1:10, y = 11:20)

df
```



```
   id  x  y
1   a  1 11
2   b  2 12
3   c  3 13
4   d  4 14
5   e  5 15
6   f  6 16
7   g  7 17
8   h  8 18
9   i  9 19
10  j 10 20
```

Useful data frame functions

- head() - shows first 6 rows
- tail() - shows last 6 rows
- dim() - returns the dimensions of data frame (i.e. number of rows and number of columns)
- nrow() - number of rows
- ncol() - number of columns
- str() - structure of data frame - name, type and preview of data in each column
- names() or colnames() - both show the names attribute for a data frame
- sapply(dataframe, class) - shows the class of each column in the data frame

More about data frames, later

UNIVERSITY OF BIRMINGHAM

v.p.lenis@bham.ac.uk (Vasilis)
l.bravo@bham.ac.uk (Laura)