Functions in R

Zina Ibrahim
zina.ibrahim@kcl.ac.uk

Outline

1. Introduction to functions in R
2. Built-in functions in R
3. Defining your own functions in R
4. Iterative structures through the apply family
5. Summary

1. Introduction to Functions

What are Functions?

- A data type called 'function'
  - Designed to perform the task you program it to.
  - Has a name which you use to call it.
  - Has arguments (list of data to go in).
  - It returns some results
- R has a massive collection of ready-to use (built-in functions).
- You can also create user-defined functions.

2. Built-in Functions in R

Built-in Functions in R

- Some numeric functions:
  - abs(x), sqrt(x), ceiling(x), cos(x), log(x), log10(x)…

```
> abs(-3)
[1] 3
```

- Some statistical functions:
  - dnorm(x), mean(x), sd(x), range(x), median(x)….

```
> vec <- c(1, 2, 3)
> mean(vec)
[1] 2
```

- Other useful functions:
  - dim(x), length(x), class(x), is.na(x), seq(x,y), rep(x,y)…

```
> length(vec)
[1] 3
```

Generic Functions

- Many R built-in functions behave differently depending on the type of arguments supplied.
- A generic function is one which may be applied to different types of inputs producing different results depending on the type of input

```
> some.chars <- c('a','b','c')
> some.nums <- c(1,2,3)
> summary(some.chars)
Length     Class      Mode
         3 character character
> summary(some.nums)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    1.0     1.5     2.0     2.0     2.5     3.0
```

Help and More Info on Built-in Functions

- `help.start()` # opens general help
- `help(foo)` # help about function foo
- `?foo` # same thing
  - `> ?mean`
- `apropos("foo")` # list all functions containing string foo
  - `> apropos("plot")`
- `example(foo)` # show an example of foo
  - `> example(summary)`
- `args(foo)` # show the arguments of foo
  - `> example(sum)`

3. Building Your Own Functions

User-defined Functions

- The great thing about R (and other scriptable languages):
  - If you don't like how a function works, you can change it by writing your own
  - If your work requires functionality not available as a built-in function, you can write your own.

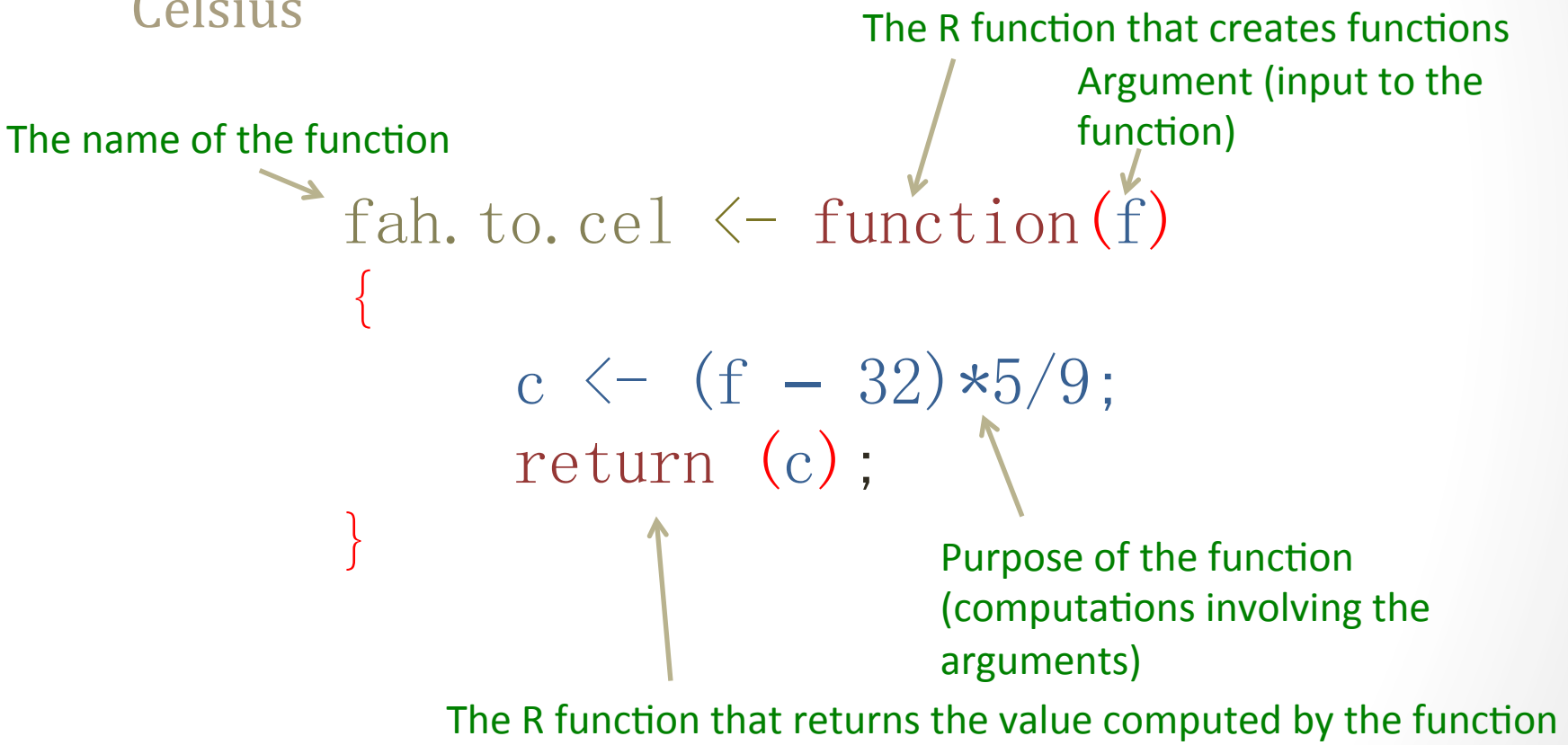
User-defined Functions

- Example: need a function to convert from Fahrenheit to Celsius

```
fah.to.cel <- function(f)
{
        c <- (f - 32)*5/9;
        return (c);
}
```

User-defined Functions

- Example: need a function to convert from Fahrenheit to Celsius



General Structure

```
function.name <- function (arg1,arg2, ...)
{
  statement 1
  statement 2
  .........

  return (value)              ##Optional
}
```

Example

- Define your own summary function:

```
full.summary <- function(x){
    print(summary(x))
    print(mean(x))
    print(sd(x))
    print(quantile(x,c(0.1,0.5,0.9)))
    par(mfrow=c(2,1))
    hist(x)
    boxplot(x,horizontal=TRUE)
}
```

- Now call it for some vector:

```
> some.vector <- c(1,2,3,4)
> full.summary(some.vector)
```

4. The Apply Family

Apply

- The functions we use are simple (e.g. sum, mean…)
- Imagine: need the mean of every column in a matrix

```
> my.matrix
[,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

  - But mean(my.matrix) will output the mean of the entire matrix!

```
> mean(my.matrix)
[1] 5
```

- We need a structured way of applying functions over the different dimensions (i.e. rows and columns) of data structures

Apply

- Solution: use the apply method!

```
> my.matrix
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

> apply(my.matrix, 1, mean)    # Row mean
[1] 4 5 6
> apply(my.matrix, 2, mean)     # Column mean
[1] 2 5 8
```

The Apply Family

- Getting a list of all apply functions is tricky

```
> apropos("apply")
 [1] ".mapply"     "apply"       "dendrapply"  "eapply"
 [5] "kernapply"   "lapply"      "mapply"      "rapply"
 [9] "sapply"      "tapply"      "vapply"
```

- Primary apply functions

	Description	Input	Output
apply	Applys a function over dimensions of a data structure	Structure with a "dimension"	A single mode structure
lapply	Applys a function to elements of a list or vector	A list or vector	A list
sapply	Applys a function to elements of a list or vector	A list or vector	A "simplified list"
tapply	Applys a function to a vector by factor(s)	A Vector + Factor(s)	Depends on # factors
by	Applys a function to a data frame by factor(s)	A Data Frame + Factor(s)	A "by" object (which is a list)
aggregate	Applys a function to columns of a data frame by factor(s)	A Data frame + Factor(s)	A Data frame

5. The Take-home Message

Why Create Functions?

- Re-use:
  - Perform the same task repeatedly with different data as input without having to change the function's code
- Modularity:
  - Your program becomes modular.
    - a number of task-specific functions that get called whenever they are needed.
  - Cleaner code!
  - You become more productive with time.
  - You can share functions!