INTERMEDIATE R

Useful Functions

Loads of useful functions

- sapply(), vapply(), lapply()
- sort()
- print()
- identical()
- ...

Mathematical utilities

```
v1 <- c(1.1, -7.1, 5.4, -2.7)
v2 <- c(-3.6, 4.1, 5.8, -8.0)
mean(c(sum(round(abs(v1))), sum(round(abs(v2)))))
```

abs()

```
v1 <- c(1.1, -7.1, 5.4, -2.7)
v2 <- c(-3.6, 4.1, 5.8, -8.0)
mean(c(sum(round(abs(v1))), sum(round(abs(v2)))))
```

```
> abs(c(1.1, -7.1, 5.4, -2.7))
[1] 1.1 7.1 5.4 2.7
> abs(c(-3.6, 4.1, 5.8, -8.0))
[1] 3.6 4.1 5.8 8.0
```

```
mean(c(sum(round(c(1.1, 7.1, 5.4, 2.7))),
       sum(round(c(3.6, 4.1, 5.8, 8.0)))))
```

round()

```
v1 <- c(1.1, -7.1, 5.4, -2.7)
v2 <- c(-3.6, 4.1, 5.8, -8.0)
mean(c(sum(round(abs(v1))), sum(round(abs(v2)))))
```

```
mean(c(sum(round(c(1.1, 7.1, 5.4, 2.7))),
       sum(round(c(3.6, 4.1, 5.8, 8.0)))))
```

```
> round(c(1.1, 7.1, 5.4, 2.7))
[1] 1 7 5 3
> round(c(3.6, 4.1, 5.8, 8.0))
[1] 4 4 6 8
```

```
mean(c(sum(c(1, 7, 5, 3)),
       sum(c(4, 4, 6, 8))))
```

sum()

```
v1 <- c(1.1, -7.1, 5.4, -2.7)
v2 <- c(-3.6, 4.1, 5.8, -8.0)
mean(c(sum(round(abs(v1))), sum(round(abs(v2)))))
```

```
mean(c(sum(c(1, 7, 5, 3)),
       sum(c(4, 4, 6, 8))))
```

```
> sum(c(1, 7, 5, 3))
[1] 16
> sum(c(4, 4, 6, 8))
[1] 22
```

```
mean(c(16, 22))
```

mean()

```
> mean(c(16, 22))
[1] 19
```

```
> v1 <- c(1.1, -7.1, 5.4, -2.7)
> v2 <- c(-3.6, 4.1, 5.8, -8.0)
> mean(c(sum(round(abs(v1))), sum(round(abs(v2)))))
[1] 19
```

Functions for data structures

```
li <- list(log = TRUE,
           ch = "hello",
           int_vec = sort(rep(seq(8, 2, by = -2), times = 2)))
```

```
sort(rep(seq(8, 2, by = -2), times = 2))
```

seq()

```
li <- list(log = TRUE,
           ch = "hello",
           int_vec = sort(rep(seq(8, 2, by = -2), times = 2)))
```

seq generates a sequence of numbers

```
sort(rep(seq(8, 2, by = -2), times = 2))
```

```
> seq(1, 10, by = 3)
[1]  1  4  7 10

> seq(8, 2, by = -2)
[1] 8 6 4 2
```

```
sort(rep(c(8, 6, 4, 2), times = 2))
```

rep()

```
li <- list(log = TRUE,
           ch = "hello",
           int_vec = sort(rep(seq(8, 2, by = -2), times = 2)))
```

```
sort(rep(c(8, 6, 4, 2), times = 2))
```

```
> rep(c(8, 6, 4, 2), times = 2)
[1] 8 6 4 2 8 6 4 2

> rep(c(8, 6, 4, 2), each = 2)
[1] 8 8 6 6 4 4 2 2
```

```
sort(c(8, 6, 4, 2, 8, 6, 4, 2))
```

sort()

```
li <- list(log = TRUE,
           ch = "hello",
           int_vec = sort(rep(seq(8, 2, by = -2), times = 2)))
```

```
> sort(c(8, 6, 4, 2, 8, 6, 4, 2))
[1] 2 2 4 4 6 6 8 8
```

a generic function

```
> sort(c(8, 6, 4, 2, 8, 6, 4, 2), decreasing = TRUE)
[1] 8 8 6 6 4 4 2 2
```

```
> sort(rep(seq(8, 2, by = -2), times = 2))
[1] 2 2 4 4 6 6 8 8
```

str()

show the contents of your data structures in a concise way

```
> li <- list(log = TRUE,
             ch = "hello",
             int_vec = sort(rep(seq(8, 2, by = -2), times = 2)))
> str(li)
List of 3
 $ log    : logi TRUE
 $ ch     : chr "hello"
 $ int_vec: num [1:8] 2 2 4 4 6 6 8 8
```

is.*(), as.*()

check the type of your data structure

```
> is.list(li)
[1] TRUE

> is.list(c(1, 2, 3))
[1] FALSE
```

convert vectors to lists

```
> li2 <- as.list(c(1, 2, 3))

> is.list(li2)
[1] TRUE

> unlist(li)
     log       ch  int_vec1  int_vec2  ... int_vec7  int_vec8
  "TRUE"  "hello"       "2"       "2"  ...      "8"       "8"
```

append(), rev()

append() allows you to add elements to a vector or a list
rev() creates a new version of li that contains the same data in the different order (reverse the list)

```
str(append(li, rev(li)))
```

```
> str(rev(li))
List of 3
 $ int_vec: num [1:8] 2 2 4 4 6 6 8 8
 $ ch     : chr "hello"
 $ log    : logi TRUE

> str(append(li, rev(li)))
List of 6
 $ log    : logi TRUE
 $ ch     : chr "hello"
 $ int_vec: num [1:8] 2 2 4 4 6 6 8 8
 $ int_vec: num [1:8] 2 2 4 4 6 6 8 8
 $ ch     : chr "hello"
 $ log    : logi TRUE
```

INTERMEDIATE R

Let’s practice!

INTERMEDIATE R

Regular Expressions

Regular Expressions

- Sequence of (meta)characters
- Pattern existence
- Pattern replacement
- Pattern extraction
- grep(), grepl()
- sub(), gsub()

It is nothing more than a sequence of characters and metacharacters that form a search pattern which you can use to match strings.
You can use a regular expression to check whether certain patterns exist in a text, to replace these patterns with other elements or to extract certain patterns out of a string.
Regexes are particularly handy when you want to clean your data. You'll often turn to regular expressions to make your data ready to further analysis, especially when you're working with data from the web or from different sources.

grepl()

```
> animals <- c("cat", "moose", "impala", "ant", "kiwi")
```

```
grepl(pattern = <regex>, x = <string>)
```

```
> grepl(pattern = "a", x = animals)
[1]  TRUE FALSE  TRUE  TRUE FALSE

> grepl(pattern = "^a", x = animals)
[1] FALSE FALSE FALSE  TRUE FALSE

> grepl(pattern = "a$", x = animals)
[1] FALSE FALSE  TRUE FALSE FALSE

> ?regex
```

With the grepl() function, we can determine, for example, which of these animals has an “a” in their names.

start with an “a”
use the caret metacharacter

end with an “a”

grep()

```
> animals <- c("cat", "moose", "impala", "ant", "kiwi")
```

```
> grepl(pattern = "a", x = animals)
[1]  TRUE FALSE  TRUE  TRUE FALSE

> grep(pattern = "a", x = animals)
[1] 1 3 4

> which(grepl(pattern = "a", x = animals))
[1] 1 3 4

> grep(pattern = "^a", x = animals)
[1] 4
```

returns a vector of indices of the elements of x that yield a match

sub(), gsub()

directly replace matches with other strings

```
> animals <- c("cat", "moose", "impala", "ant", "kiwi")
```

```
sub(pattern = <regex>, replacement = <str>, x = <str>)
```

```
> sub(pattern = "a", replacement = "o", x = animals)
[1] "cot"    "moose"  "impola" "ont"    "kiwi"
```

sub() function only looks for the first match in the string

```
> gsub(pattern = "a", replacement = "o", x = animals)
[1] "cot"    "moose"  "impolo" "ont"    "kiwi"
```

replace every single match of a pattern in a string with the replacement argument

sub(), gsub()

```
> animals <- c("cat", "moose", "impala", "ant", "kiwi")
```

```
> sub(pattern = "a", replacement = "o", x = animals)
[1] "cot"    "moose"  "impola" "ont"    "kiwi"

> gsub(pattern = "a", replacement = "o", x = animals)
[1] "cot"    "moose"  "impolo" "ont"    "kiwi"
```

```
> gsub(pattern = "a|i", replacement = "_", x = animals)
[1] "c_t"    "moose"  "_mp_l_" "_nt"    "k_w_"

> gsub(pattern = "a|i|o", replacement = "_", x = animals)
[1] "c_t"    "m__se"  "_mp_l_" "_nt"    "k_w_"
```

INTERMEDIATE R

Let’s practice!

INTERMEDIATE R

Times & Dates

Today, right now!

```
> today <- Sys.Date()
> today
[1] "2015-05-07"

> class(today)
[1] "Date"
```

```
> now <- Sys.time()
> now
[1] "2015-05-07 10:34:52 CEST"

> class(now)
[1] "POSIXct" "POSIXt"
```

Create Date objects

```
> my_date <- as.Date("1971-05-14")
> my_date
[1] "1971-05-14"

> class(my_date)
[1] "Date"

> my_date <- as.Date("1971-14-05")
Error in charToDate(x) :
  character string is not in a standard unambiguous format

> my_date <- as.Date("1971-14-05", format = "%Y-%d-%m")
> my_date
[1] "1971-05-14"
```

Default format
"%Y-%m-%d"

%Y = 4-digit year
%m = 2-digit month
%d = 2-digit day

Create POSIXct objects

```
> my_time <- as.POSIXct("1971-05-14 11:25:15")
> my_time
[1] "1971-05-14 11:25:15 CET"
```

Date arithmetic

```
> my_date
[1] "1971-05-14"
                    days incremented by 1
> my_date + 1
[1] "1971-05-15"

> my_date2 <- as.Date("1998-09-29")

> my_date2 - my_date
Time difference of 10000 days
```

POSIXct arithmetic

```
> my_time
[1] "1971-05-14 11:25:15 CET"

> my_time + 1
[1] "1971-05-14 11:25:16 CET"

> my_time2 <- as.POSIXct("1974-07-14 21:11:55 CET")

> my_time2 - my_time
Time difference of 1157.407 days
```

seconds incremented by 1

For any time difference, R will automatically display an easily interpretable time difference.

Under the hood

```
> my_date
[1] "1971-05-14"

> unclass(my_date)
[1] 498
```

498 days from January 1, 1970

convert Date to numeric

```
> my_time
[1] "1971-05-14 11:25:15 CET"

> unclass(my_time)
[1] 43064715
attr(,"tzone")
[1] ""
```

>43MM seconds from January 1, 1970, 00:00:00

Dedicated R Packages

- lubridate
- zoo
- xts

INTERMEDIATE R

Let’s practice!