Vectors, lists and iterators

Vectors

- A **vector** is a sequence of elements of the same type.
- A **list** is a recursive vector
- “type” of vectors = `integer`, `double`, `character`, `logical`, `factor`, or `date`.
- 6 types of atomic vectors =
- `numeric` is used to represent vectors that are either integers or doubles.
- Use `c()` to create vectors.
- Use `typeof()` to see the type of vector and `is_*()` to check the type of vector.
- Note: factors are usually integers w/ extra attributes. DAtes are doubles with extra attributes.

Vector basics

```
# each element can havea name
    x <- c(horse = 7, man = 1, dog = 8)
# you can see it w/ names function
names(x)
```

```
[1] ”horse” ”man”   ”dog”
```

```
# subset with brackets
    x <- c(”I”, ”like”, ”dogs”)
    x[2:3]
```

```
[1] "like" "dogs"
```

```
lvec <- c(TRUE, FALSE, TRUE)
x[lvec]
```

```
[1] "I"    "dogs"
```

```
x <- c(horse = 7, man = 1, dog = 8)
x["man"]
```

```
man
  1
```

Lists

- **Lists are vectors** whose elements can be of different types.
- Two brackets `[[` only returns a single elements and drops the name.

```
x[3]
```

```
dog
  8
```

```
x[[3]]
```

```
[1] 8
```

```
names <- c('john', 'Mark', 'sam')
numbers <- c(1, 2, 3)
letters <- c('A', 'B', 'C')

full_list <- list(names, numbers, letters)
```

```
full_list[1] # first list
```

```
[[1]]
[1] "john" "Mark" "sam"
```

```
full_list[[1]] # also first lsit
```

```
[1] "john" "Mark" "sam"
```

```
full_list[[2]][3] # third item of second list
```

```
[1] 3
```

```
# extract the number "4"
L2 <- list(T, 12, 0.5, 'cat', list(2:5))
L2[[5]][[1]][3] # fifth item in list L2, first list in list(2:5), third item
```

```
[1] 4
```

```
# extract "A"
L2 <- list(T, 12, 0.5, 'cat', list(2:5, c('R', 'A')))
L2[[5]][[2]][2]
```

```
[1] "A"
```

- Use dollar signs $ (just like in data frames) to extract named list elements.

```
# my_first_list$z
```

- You can remove elements of a list by substituting them with NULL.

```
# str(my_first_list)
# my_first_list$x <- NULL
```

```
    # str(my_first_list)
```

Examples

Use and show R code that will extract “Dolphins” from the vector shown below

```
V <- c(”Bears”, ”Lions”, ”Dolphins”, ”Eagles”, ”Bengals”)
V[3]                    # select tiems
```

```
[1] ”Dolphins”
```

```
V[c(F, F, T, F, F)]    # using logical values
```

```
[1] ”Dolphins”
```

```
V[c(-1, -2, -4, -5)]   # using negative numbers
```

```
[1] ”Dolphins”
```

Number of elements in each part of list

```
K <- list(x = 3:7, ”never”, 43, y = list(10,20,30))
purrr::map_int(K, length)
```

```
x     y
5 1 1 3
```

For loops

- *Iteration* is the repetition of some amount of code.

```
x <- c(8, 1, 3, 1, 3)
```

```r
# loop to total w/o "sum" function
sumval <- 0
    for (i in seq_along(x)) {
       sumval <- sumval + x[[i]]
    }
    sumval
```

```
[1] 16
```

```r
# same as doing this:
    sumval <- 0
    sumval <- sumval + x[[1]]
    sumval <- sumval + x[[2]]
    sumval <- sumval + x[[3]]
    sumval <- sumval + x[[4]]
    sumval <- sumval + x[[5]]
    sumval
```

```
[1] 16
```

- Each for loop contains the following elements:

    1. **Output**: This is `sumval` above. We allocate the space for the output *before* the for loop.
    2. **Sequence**: This is `seq_along(x)` above, which evaluates to `1 2 3 4 5`. These are the values that `i` will go through each iteration.
    3. **Body**: This is the code between the curly braces `{}`. This is the code that will be evaluated each iteration with a new value of `i`.

- You often want to fill a vector with values. You should create this vector beforehand using the `vector()` function.
- For example, let's calculate a vector of cumulative sums of `x`.

```r
    cumvec <- vector(mode = "double", length = length(x))
    cumvec
```

```
[1] 0 0 0 0 0
```

```
    for (i in seq_along(cumvec)) {
       if (i == 1) {
         cumvec[[i]] <- x[[i]]
       } else {
         cumvec[[i]] <- cumvec[[i - 1]] + x[[i]]
       }
    }
    cumvec
```

```
[1]  8  9 12 13 16
```

```
    ## Same as cumsum(x)
    cumsum(x)
```

```
[1]  8  9 12 13 16
```

- Looping is often done over the columns of a data frame.
- Note: for a data frame `df`, `seq_along(df)` is the same as `1:ncol(df)` which is the same as `1:length(df)` (since data frames are special cases of lists).
- Let's calculate the mean of each column of `mtcars`

```
    data("mtcars")
    mean_vec <- vector(mode = "numeric", length = length(mtcars))
    for (i in seq_along(mtcars)) {
      mean_vec[[i]] <- mean(mtcars[[i]], na.rm = TRUE)
    }
    mean_vec
```

```
 [1]  20.090625   6.187500 230.721875 146.687500   3.596563   3.217250
 [7]  17.848750   0.437500   0.406250   3.687500   2.812500
```

```
    colMeans(mtcars)
```

```
      mpg       cyl      disp        hp      drat        wt      qsec
20.090625  6.187500 230.721875 146.687500  3.596563  3.217250 17.848750
       vs        am      gear      carb
 0.437500  0.406250  3.687500  2.812500
```

- **Exercise**: Use a for loop to calculate the standard deviation of each penguin trait in the `penguins` data frame from the `palmerpenguins` package.

```
library(palmerpenguins)
 data("penguins")
 sdvec <- rep(NA, length = length(penguins))
 for (i in seq_along(penguins)) {
   if (is.numeric(penguins[[i]])) {
     sdvec[[i]] <- sd(penguins[[i]], na.rm = TRUE)
   }
 }
 sdvec
```

```
[1]          NA          NA   5.4595837   1.9747932  14.0617137 801.9545357
[7]          NA   0.8183559
```

Purrr

Basics

- R is a functional programming language. Which means that you can pass functions to functions.

```
library(purrr)
```

- Suppose on `mtcars` we want to calculate the column-wise mean, the column-wise median, the column-wise standard deviation, the column-wise maximum, the column-wise minimum, and the column-wise MAD.
- Ideally, we would like to just tell R what function to apply to each column of `mtcars`. This is what the purrr package allows us to do.
- purrr is a part of the tidyverse, and so does not need to be loaded separately.
- `map_*()` takes a vector (or list or data frame) as input, applies a provided function on each element of that vector, and outputs a vector of the same length.
  - `map()` returns a list.
  - `map_lgl()` returns a logical vector.
  - `map_int()` returns an integer vector.
  - `map_dbl()` returns a double vector.
  - `map_chr()` returns a character vector.

This is how we would do this with a for loop

```
  funvec <- rep(NA, length = length(mtcars))
      for (i in seq_along(funvec)) {
        funvec[i] <- median(mtcars[[i]], na.rm = TRUE)
      }
      funvec
```

```
 [1]  19.200   6.000 196.300 123.000   3.695   3.325  17.710   0.000   0.000
[10]   4.000   2.000
```

This is how we would do this with purrr

```
      map_dbl(mtcars, mean)
```

```
      mpg       cyl      disp        hp      drat        wt      qsec
 20.090625  6.187500 230.721875 146.687500  3.596563  3.217250 17.848750
       vs        am      gear      carb
  0.437500  0.406250  3.687500  2.812500
```

```
      map_dbl(mtcars, median)
```

```
   mpg     cyl    disp      hp    drat      wt    qsec      vs      am    gear
 19.200   6.000 196.300 123.000   3.695   3.325  17.710   0.000   0.000   4.000
  carb
 2.000
```

```
      map_dbl(mtcars, sd)
```

```
      mpg       cyl      disp        hp      drat        wt
 6.0269481  1.7859216 123.9386938 68.5628685  0.5346787  0.9784574
     qsec        vs        am      gear      carb
 1.7869432  0.5040161  0.4989909  0.7378041  1.6152000
```

```
      map_dbl(mtcars, mad)
```

```
       mpg         cyl        disp          hp        drat          wt
 5.4114900   2.9652000 140.4763500  77.0952000   0.7042350   0.7672455
      qsec          vs          am        gear        carb
 1.4158830   0.0000000   0.0000000   1.4826000   1.4826000
```

```
map_dbl(mtcars, min)
```

```
   mpg    cyl   disp     hp   drat     wt   qsec     vs     am   gear   carb
10.400  4.000 71.100 52.000  2.760  1.513 14.500  0.000  0.000  3.000  1.000
```

```
map_dbl(mtcars, max)
```

```
    mpg     cyl    disp      hp    drat      wt    qsec      vs      am    gear
 33.900   8.000 472.000 335.000   4.930   5.424  22.900   1.000   1.000   5.000
   carb
  8.000
```

```
# getting the summary
map(mtcars, summary)
```

```
$mpg
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  10.40   15.43   19.20   20.09   22.80   33.90

$cyl
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  4.000   4.000   6.000   6.188   8.000   8.000

$disp
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   71.1   120.8   196.3   230.7   326.0   472.0

$hp
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   52.0    96.5   123.0   146.7   180.0   335.0

$drat
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
```

```
  2.760   3.080   3.695   3.597   3.920   4.930

$wt
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.513   2.581   3.325   3.217   3.610   5.424

$qsec
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  14.50   16.89   17.71   17.85   18.90   22.90

$vs
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.0000  0.0000  0.4375  1.0000  1.0000

$am
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.0000  0.0000  0.4062  1.0000  1.0000

$gear
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  3.000   3.000   4.000   3.688   4.000   5.000

$carb
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   2.000   2.000   2.812   4.000   8.000
```

```
# determining type of column
library(nycflights13)
map_chr(flights, typeof)
```

```
        year          month            day       dep_time sched_dep_time
   "integer"      "integer"      "integer"      "integer"      "integer"
   dep_delay       arr_time sched_arr_time      arr_delay        carrier
    "double"      "integer"      "integer"       "double"    "character"
      flight        tailnum         origin           dest       air_time
   "integer"    "character"    "character"    "character"       "double"
    distance           hour         minute      time_hour
    "double"       "double"       "double"       "double"
```

```
    # number of unique values
    map_int(penguins, function(x) length(unique(x)))
```

```
          species            island    bill_length_mm     bill_depth_mm
                3                 3               165                81
flipper_length_mm       body_mass_g               sex              year
               56                95                 3                 3
```

```
    # 10 random normal distribution
    map_dfc(seq(-10, 100, by = 10), rnorm, n = 10) -> temp
```

```
New names:
* `` -> `...1`
* `` -> `...2`
* `` -> `...3`
* `` -> `...4`
* `` -> `...5`
* `` -> `...6`
* `` -> `...7`
* `` -> `...8`
* `` -> `...9`
* `` -> `...10`
* `` -> `...11`
* `` -> `...12`
```

Shortcuts

- You can refer to elements of the vector by "." in a `map()` call if the `.f` argument is preceded by a "~". For example, the following are three equivalent ways to calculate the mean of each column in `mtcars`.

```
    map_dbl(mtcars, mean)
```

```
      mpg       cyl      disp        hp      drat        wt      qsec
20.090625  6.187500 230.721875 146.687500  3.596563  3.217250 17.848750
       vs        am      gear      carb
 0.437500  0.406250  3.687500  2.812500
```

```
    map_dbl(mtcars, function(.) mean(.))
```

```
      mpg       cyl      disp        hp      drat        wt      qsec
20.090625  6.187500 230.721875 146.687500  3.596563  3.217250 17.848750
       vs        am      gear      carb
 0.437500  0.406250  3.687500  2.812500
```

```
    map_dbl(mtcars, ~mean(.))
```

```
      mpg       cyl      disp        hp      drat        wt      qsec
20.090625  6.187500 230.721875 146.687500  3.596563  3.217250 17.848750
       vs        am      gear      carb
 0.437500  0.406250  3.687500  2.812500
```

Tidyverse allows for some shortcuts

```
    mtcars %>%                              # mt cars data
      split(.$cyl) %>%                      # group by cyl
      map(~t.test(mpg ~ am, data = .)) %>%  # t-test of difference bw mpg & am
      map(~.$p.value)                       # for each t-test extract the pval
```

```
$`4`
[1] 0.01801712

$`6`
[1] 0.187123

$`8`
[1] 0.7038727
```

- `keep()` selects all variables that return `TRUE` according to some function.
- E.g. let's keep all numeric variables and calculate their means in the `palmerpenguins::penguins` data frame.

```
library(palmerpenguins)
    data("penguins")
    penguins %>%
      keep(is.numeric) %>%
```

```
        map_dbl(mean, na.rm = TRUE)
```

```
   bill_length_mm     bill_depth_mm flipper_length_mm       body_mass_g
         43.92193          17.15117         200.91520        4201.75439
             year
      2008.02907
```

- `discard()` will select all variables that return `FALSE` according to some function.
- Let's count the number of each value for each categorical variable:

```
 penguins %>%
      discard(is.numeric) %>%   # drop all numeric columns
      map(table)                # make table of all character columns
```

```
$species

   Adelie Chinstrap    Gentoo
      152        68       124

$island

   Biscoe     Dream Torgersen
      168       124        52

$sex

female   male
   165    168
```

- In the `mtcars` data frame, keep only variables that have a mean greater than `10` and calculate their mean. Hint: You'll have to use some of the shortcuts above.

```
mtcars %>%
      keep(~mean(.) > 10) %>%
      map(mean)
```

```
$mpg
[1] 20.09062

$disp
```

```
[1] 230.7219

$hp
[1] 146.6875

$qsec
[1] 17.84875
```