# Data Structures

# Agenda

- **Vectors**
- Matrices
- Arrays
- Lists
- **Data Frames (and Tibbles)**
- Structures of structures

# Vectors

# First data structure: vectors

Group related data values into one object, a **data structure**
A **vector** is a sequence of values, all of the same type

```
x <- c(7, 8, 10, 45)
x
```

```
## [1]  7  8 10 45
```

```
is.vector(x)
```

```
## [1] TRUE
```

`c()` function returns a vector containing all its arguments in order
`x[1]` is the first element, `x[4]` is the 4th element
`x[-4]` is a vector containing all but the fourth element

# Vectors cont'd.

`vector(length=6)` returns an empty vector of length 6; helpful for filling things up later

```
weekly.hours <- vector(length=5)
weekly.hours[5] <- 8
```

# Vector arithmetic

Operators apply to vectors "pairwise" or "elementwise":

```
y <- c(-7, -8, -10, -45)
x+y
```

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

```
x*y
```

```
## [1]   -49   -64  -100 -2025
```

# Recycling

**Recycling** repeats elements in shorter vector when combined with longer

```
x + c(-7,-8)
```

```
## [1]  0  0  3 37
```

```
x^c(1,0,-1,0.5)
```

```
## [1] 7.000000 1.000000 0.100000 6.708204
```

Single numbers are vectors of length 1 for purposes of recycling:

```
2*x
```

```
## [1] 14 16 20 90
```

# Comparison functions

Can also do pairwise comparisons:

```
x > 9
```

```
## [1] FALSE FALSE  TRUE  TRUE
```

Note: returns Boolean vector
Boolean operators work elementwise:

```
(x > 9) & (x < 20)
```

```
## [1] FALSE FALSE  TRUE FALSE
```

# Comparison functions

To compare whole vectors, best to use `identical()` or `all.equal()`:

```
x == -y
```

```
## [1] TRUE TRUE TRUE TRUE
```

```
identical(x,-y)
```

```
## [1] TRUE
```

```
identical(c(0.5-0.3,0.3-0.1),c(0.3-0.1,0.5-0.3))
```

```
## [1] FALSE
```

```
all.equal(c(0.5-0.3,0.3-0.1),c(0.3-0.1,0.5-0.3))
```

```
## [1] TRUE
```

# Functions on vectors

Lots of functions take vectors as arguments:

- `mean()`, `median()`, `sd()`, `var()`, `max()`, `min()`, `length()`, `sum()`: return single numbers
- `sort()` returns a new vector
- `hist()` takes a vector of numbers and produces a histogram, a highly structured object, with the side-effect of making a plot
- Similarly `ecdf()` produces a cumulative-density-function object
- `summary()` gives a five-number summary of numerical vectors
- `any()` and `all()` are useful on Boolean vectors

# Addressing vectors

Vector of indices:

```
x;x[2];x[4]
```

```
## [1]  7  8 10 45
```

```
## [1] 8
```

```
## [1] 45
```

```
x[c(2,4)]
```

```
## [1]  8 45
```

Vector of negative indices

```
x[c(-1,-3)]
```

```
## [1]  8 45
```

(why that, and not `8 10`?)

# Addressing vectors cont'd.

Boolean vector:

```
x[x>9]
```

```
## [1] 10 45
```

```
y[x>9]
```

```
## [1] -10 -45
```

`which()` turns a Boolean vector in vector of TRUE indices:

```
places <- which(x > 9)
places
```

```
## [1] 3 4
```

```
y[places]
```

```
## [1] -10 -45
```

# Matrices, Arrays and Lists

# Vector structures, starting with arrays

Many data structures in R are made by adding bells and whistles to vectors, so "vector structures"

A **matrix** in R is a collections of homogeneous elements arranged in 2 dimensions

```
matrix(1:15, nrow = 4)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    5    9   13
## [2,]    2    6   10   14
## [3,]    3    7   11   15
## [4,]    4    8   12    1
```

# Arrays

**arrays** are basically matrices in higher dimensions

```
x <- c(7, 8, 10, 45 , 70, 80 , 100, 250)
x.arr <- array(x,dim=c(2,2,2))
x.arr
```

```
## , , 1
##
##      [,1] [,2]
## [1,]    7   10
## [2,]    8   45
##
## , , 2
##
##      [,1] [,2]
## [1,]   70  100
## [2,]   80  250
```

- `dim` says how many rows and columns; filled by columns
- Can have $3, 4, \ldots n$ dimensional arrays; `dim` is a length-$n$ vector

# Lists

Sequence of values, *not* necessarily all of the same type

```
my.distribution <- list("exponential",7,FALSE)
my.distribution
```

```
## [[1]]
## [1] "exponential"
##
## [[2]]
## [1] 7
##
## [[3]]
## [1] FALSE
```

Most of what you can do with vectors you can also do with lists

# Dataframes and Tibbles

# Dataframes

- Dataframe = the classic data table, $n$ rows for cases, $p$ columns for variables
- Not just a matrix because *columns can have different types*
- Many matrix functions also work for dataframes (`rowSums()`, `summary()`, `apply()`)

but no matrix multiplication of dataframes, even if all columns are numeric

# Dataframes

- 2D tables of data
- Each case/unit is a row
- Each variable is a column
- Variables can be of any type (numbers, text, Booleans, ...)
- Both rows and columns can get names

# An example dataframe

```
mtcars ##already in R environment
```

```
##                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL        17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC       15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood 10.4  8 472.0 205 2.93 5.250 17.98  0  0    3    4
```

## Column names are preserved or guessed if not explicitly set

```
colnames(mtcars)
```

```
##  [1] "mpg"   "cyl"   "disp"  "hp"    "drat"  "wt"    "qsec"  "vs"    "am"
## [10] "gear"  "carb"  "watts"
```

```
mtcars[1,]
```

```
##           mpg cyl disp  hp drat   wt  qsec vs am gear carb watts
## Mazda RX4  21   6  160 110  3.9 2.62 16.46  0  1    4    4 82027
```

# Dataframe access

- By row and column index

```
mtcars[10,3]
```

```
## [1] 167.6
```

- By row and column names

```
mtcars["Honda Civic","mpg"]
```

```
## [1] 30.4
```

# Dataframe access (cont'd)

- All of a row:

```
mtcars["Toyota Corolla",]
```

```
##                 mpg cyl disp hp drat    wt qsec vs am gear carb   watts
## Toyota Corolla 33.9   4 71.1 65 4.22 1.835 19.9  1  1    4    1 48470.5
```

# Dataframe access (cont'd.)

- All of a column:

```
mtcars[,1]
mtcars[,"mpg"]
mtcars$mpg
```

# Dataframe access (cont'd.)

- Rows matching a condition:

```
mtcars[mtcars$mpg>19, "cyl"]
```

```
##  [1] 6 6 4 6 4 4 6 4 4 4 4 8 4 4 4 6 4
```

```
mtcars[mtcars$cyl==4, "mpg"]
```

```
##  [1] 22.8 24.4 22.8 32.4 30.4 33.9 21.5 27.3 26.0 30.4 21.4
```

# Adding rows and columns

We can add rows or columns to an array or data-frame with `rbind()` and `cbind()`, but be careful about forced type conversions

```
a.data.frame()
rbind(a.data.frame,list(v1=-3,v2=-5,logicals=TRUE))
rbind(a.data.frame,c(3,4,6))
```

# Another way to add (or replace) a column

```
mtcars$watts <- mtcars$hp * 745.7
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb    watts
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4  82027.0
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4  82027.0
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1  69350.1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1  82027.0
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2 130497.5
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1  78298.5
```

*Internally, a dataframe is basically a list of vectors

# Tibbles

Tibbles are a modern take on data frames. They keep the features that have stood the test of time, and drop the features that used to be convenient but are now frustrating (i.e. converting character vectors to factors).

```
library(tibble)
tibble(x = 1:5, y = x ^ 2)
```

```
## # A tibble: 5 x 2
##       x     y
##   <int> <dbl>
## 1     1     1
## 2     2     4
## 3     3     9
## 4     4    16
## 5     5    25
```

https://cran.r-project.org/web/packages/tibble

## Summary

- Matrices act like you'd hope they would
- Arrays add multi-dimensional structure to vectors
- Lists let us combine different types of data
- Dataframes are hybrids of matrices and lists, for classic tabular data

## References

- http://www.stat.cmu.edu/~cshalizi/statcomp/
- https://www.r-project.org/
- https://www.rstudio.com/