Everything is a vector

Vectors are built with the `c()` function (for “combine”)

```r
a <- c(10, 20, 30, 40)
```

```r
a
```

```r
## [1] 10 20 30 40
```

Vectors are 1-indexed (like Matlab, Fortran, or Julia)

```r
a[1]
```

```r
## [1] 10
```

```r
a[2:4]
```

```r
## [1] 20 30 40
```
For historical reasons, R assigns objects to variables using `<-` (said “gets”)

```r
a <- c(10, 20, 30, 40)
```

However, the equals sign also works

```r
a = c(11, 22, 33, 44)
```

Also, periods are allowed in variables names

```r
a.var.name <- 7
is.na(a.var.name)
```

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

It’s best to avoid this and use underscores (_) instead
Vector Types

Numeric

```r
r <- rnorm(4)
r

## [1]  1.3493629 -0.2646543  1.9097851  0.4067838
```

Logical (TRUE and FALSE can be abbreviated T and F)

```r
r < 0.0

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

Character

```r
c <- c("this", "is", "a", "character")
```
Notes about characters

Everything is a vector of “strings”; there are no individual “letters”

```r
s <- "a string"
length(s)
```

```r
## [1] 1
```

```r
s[1]
```

```r
## [1] "a string"
```

```r
s[2]
```

```r
## [1] NA
```
Notes about characters (continued)

```
nchar("a string")
## [1] 8

c <- c("this", "is", "a", "character")
length(c)
## [1] 4

nchar(c)
## [1] 4 2 1 9```
Everything in a vector is the same type

R will convert objects into the same type

```r
c(1, 2, "three")
```

```r
# [1] "1"   "2"   "three"
```

For mixed types, use a list

```r
list(1, 2, "three")
```

```r
# [[1]]
# [1] 1
#
# [[2]]
# [1] 2
#
# [[3]]
# [1] "three"
```
Factors are special vectors

Factors look like regular vectors

```r
x1 <- as.factor(c("wt", "ko", "wt"))
x1
```

```r
# [1] wt ko wt
# Levels: ko wt
```
Factors are special vectors

Factors look like regular vectors

```r
x1 <- as.factor(c("wt", "ko", "wt"))
x1
```

```
## [1] wt ko wt
## Levels: ko wt
```

But they are stored as integers and a “key” of levels

```r
as.integer(x1)
```

```
## [1] 2 1 2
```

```r
levels(x1)
```

```
## [1] "ko" "wt"
```
Functions

Most functions are *vectorized* and operate elementwise

```r
a <- runif(4)
a
## [1] 0.7240759 0.7865879 0.2458798 0.1849558
10*a + 1
## [1] 8.240759 8.865879 3.458798 2.849558
sqrt(a) + cos(a)
## [1] 1.600038 1.593163 1.465786 1.413009
```

You can always ask for help: ?sqrt.
Data Frames

Data frames hold tables of data.

data <- read.csv("MonkeyThrow.csv")
data

## run hand hat boots distance
## 1 5 left yes yes 4.5
## 2 6 right yes yes 6.0
## 3 2 left no yes 7.0
## 4 7 right no yes 9.5
## 5 1 left yes no 5.0
## 6 8 right yes no 6.5
## 7 4 left no no 4.0
## 8 3 right no no 7.5
The length of a data frame is the number of *columns*.
What are data frames?

```r
head(data, n=5) # just the first 5 rows
```

```
##     run hand hat boots distance
## 1 5  left yes yes 4.5
## 2 6  right yes yes 6.0
## 3 2  left no yes 7.0
## 4 7  right no yes 9.5
## 5 1  left yes no 5.0
```

- Each column in a data frame is a vector (and must be the same type).
- Each row contains data of different types
What types ("classes") of data are in our data frame?

```r
lapply(data, class)  # applies `class` to each column
```

```r
## $run
## [1] "integer"
##
## $hand
## [1] "factor"
##
## $hat
## [1] "factor"
##
## $boots
## [1] "factor"
##
## $distance
## [1] "numeric"
```
Working with columns

data$hat

## [1] yes yes no  no  yes  yes  no  no  no
## Levels: no yes

log10(data$distance)

## [1] 0.6532125 0.7781513 0.8450980 0.9777236 0.6989700 0.8129134 0.6020600
## [8] 0.8750613
Making new columns

```r
data$log_dist <- log10(data$distance)

data

# run  hand  hat  boots distance  log_dist
# 1   5  left  yes  yes   4.5   0.6532125
# 2   6  right yes  yes   6.0   0.7781513
# 3   2  left  no   yes   7.0   0.8450980
# 4   7  right no   yes   9.5   0.9777236
# 5   1  left  yes  no    5.0   0.6989700
# 6   8  right yes  no    6.5   0.8129134
# 7   4  left  no  no    4.0   0.6020600
# 8   3  right no  no    7.5   0.8750613
```
Nevermind, let’s delete that column

data$log_dist <- NULL
data

##             run hand hat boots distance
## 1      5 left yes yes  4.5
## 2      6 right yes yes  6.0
## 3      2 left  no yes  7.0
## 4      7 right  no yes  9.5
## 5      1 left yes  no  5.0
## 6      8 right yes  no  6.5
## 7      4 left  no  no  4.0
## 8      3 right  no  no  7.5
Visualizing data

```r
plot(data$run, data$distance)
```
We can do better

```r
plot(data$run, data$distance, xlab="run number", ylab="distance", cex=2, pch=20)
```
Many of the functions we use are in libraries. We need to load the library first.

```r
library("doetools") # include `farplot` function
farplot(data, response="distance", factors=c("hand", "hat", "boots"))
```