# R_intro_2


Karel Fišer, 2016

# Last intro

R

```
>
> x <- 2
> x <- c(1, 2, 3)
> x[2]
> 5 + x
> mean(x)
> q()
```

# Variables – content types

```
> x <- c(2, 10, 3) # numeric
> y <- c("ball", "dog") # character
> z <- c(TRUE, TRUE) # logical
> factor_x <- factor(x) # factor (categorical)
>
> # or integer
> # … mind the coercion logical > integer >
  numeric > caracter
```

```
# factor is integer vector with attributes
```

# Variables – dimensions

**1D** – atomic vector, list

**2D** – matrix, data frame

nD – array

# Variables – homogeneity

**homogenous** – vector, matrix, arrays

**heterogenous** – list, data frame

# Variable constructors

```
> # atomic vector, same types, 1d:
> x <- c(2, 10, 3)

> # matrix, same column lengths, same types, 2d:
> m <- matrix(1:20, nrow=5,ncol=4)

> # data frame, same column lengths, 2d:
> d <- data.frame(c("blue", "red"), c(4, 8))

> # list, 1d (list is a vector, but possibly
  branching):
> l <- list(3, c(TRUE, FALSE))
```

# Rectangular data

Pretty much all of the data we deal with is rectangular: has columns and rows.

Statistically, we want variables and observations.

Variables go in columns, observations go in rows (with occasional exceptions)

**Hadley Wickham**

# Your turn

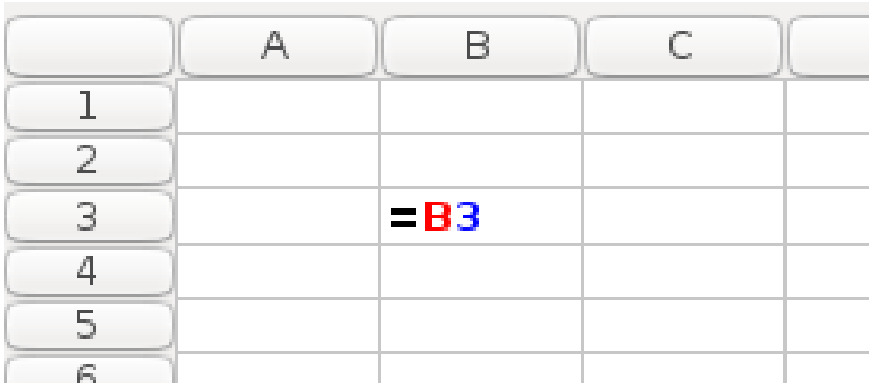
create variables of all types

# Indexing

```
> x[1]
> x[3:6]
```

```
  m[rows , columns]
```

```
> m[3, 2]
> m[1, 2:4]
> m[1, c(2,4)]
> m[2, ]
> m[,-3]
```



# Named variables

```
> x <- c(a=2, b=10, c=3)
> names(x)
```

```
> d <- data.frame(colour=c("blue", "red"),
  legs=c(4, 8))
> colnames(d)
> rownames(d)
> rownames(d) <- c("cat", "spider")
> d
```

# Accessing variables by names

```
> d[, 2]
> d[, "legs"]
> d["cat",]
```

access column by **d$column_name** notation

```
> d$legs # only for columns
> d$legs[2]
```

# Indexing list

```
> goulash <- list(1, second=2, "dog", 4,
  c(50, 51), 3:7, last=list(three=c(1, 2,
  3), veggie="Paprika"))

> str(goulash)

> goulash[5]    # returns list
> goulash[[5]]  # returns list content
> goulash[[5]][1]
> goulash$last$three[2]
```

# Your turn

- sum third element of x with
  first value in second column of d

- same using names

# Exploring variables

```
> length(x)


> d2 <- data.frame(a=1:1000, b=1000:1, c=11:1010)
> dim(d2)
> str(d2)
> head(d2)
> tail(d2, n=1)


> class(x)
> is.data.frame(x)
```

# Altering and combining

```
> rownames(d) <- c("cat", "spider")
> d[2, 2] <- 7
> d[,1] <- c("ginger", "black")


> lives <- c(9, 1)
> d[, 3] <- lives
> d <- data.frame(d, lives)
> d <- cbind(d, lives)
```

# Variable manipulation

```
> t(m)
> c(x, 2:5); cbind(x, z); rbind(x, z)
> rbind(m[1,], x) # repeating shorter vector!

> z <- c(1, 2, 3)
> class(z)
> z <- as.character(z)
> class(z)

> # missing values are NA in R
> data[data=="NotDetermined"] <- NA
```

# Your turn

"Experiment performed on 1.1.1900. Six measurements of two  variables: width and height. Both in mm."

- prepare variable containing description of an experiment and measured values.

Hint: e.g. list.

# iris dataset

```
> ?iris
```

This famous (Fisher's or Anderson's) iris data set gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are Iris setosa, versicolor, and virginica.

```
> data() # datasets in R
> str(iris)
> data <- iris
```

# Ordering

```
> data$Sepal.Length
> order(data$Sepal.Length) # indexes of ordered vector
> data$Sepal.Length[order(data$Sepal.Length)] #
  ordered vector
> data[order(data$Sepal.Length),] # ordered data.frame
```

```
> # sort by Sepal.Length (ascending) and Sepal.Width
  (descending):
> data[order(data$Sepal.Length, -data$Sepal.Width),]
```

# Variable manipulation

```
> # change col order:
> data <- data[, c(5, 1:4)]


> # new vector based on two columns:
> Sepal_square <-
  data$Sepal.Length*data$Sepal.Width


> # add new variable/col to the data:
> data <- cbind(data, Sepal_square)
```

# Finding patterns

```
> unique(data$Species)

> # find pattern "virginica":
> which(data$Species=="virginica")
> grep("virginica", data$Species)

> # find and replace:
> gsub("virginica", "sp.", data$Species)
> data$Species <- gsub("virginica", "sp.",
  data$Species)
```

# Your turn

- make "Petal_square" column and sort data on it

- change all "setosa" to "set" and "versicolor" to "ver"

# Genes in human genome

get all genes in org.Hs.eg.db

```
> library(Homo.sapiens)
> genes <- select(Homo.sapiens, keys =
  keys(Homo.sapiens, keytype = "GENENAME"),
  columns = c("SYMBOL","TXCHROM"), keytype =
  "GENENAME")
> dim(genes)
[1] 61117     3
```

# Kinases in human genome

select genes with “kinase” in their name

```
> kinases <- genes[grep("kinase",
  genes$GENENAME), ]
> dim(kinases)
[1] 977   3
```

# Kinases on chromosomes

how many kinases on each chromosome

```
> kinases_on_chr <- table(kinases$TXCHROM)
> kinases_on_chr[order(kinases_on_chr)]
> tail(kinases_on_chr[order(kinases_on_chr)],
  5)
```

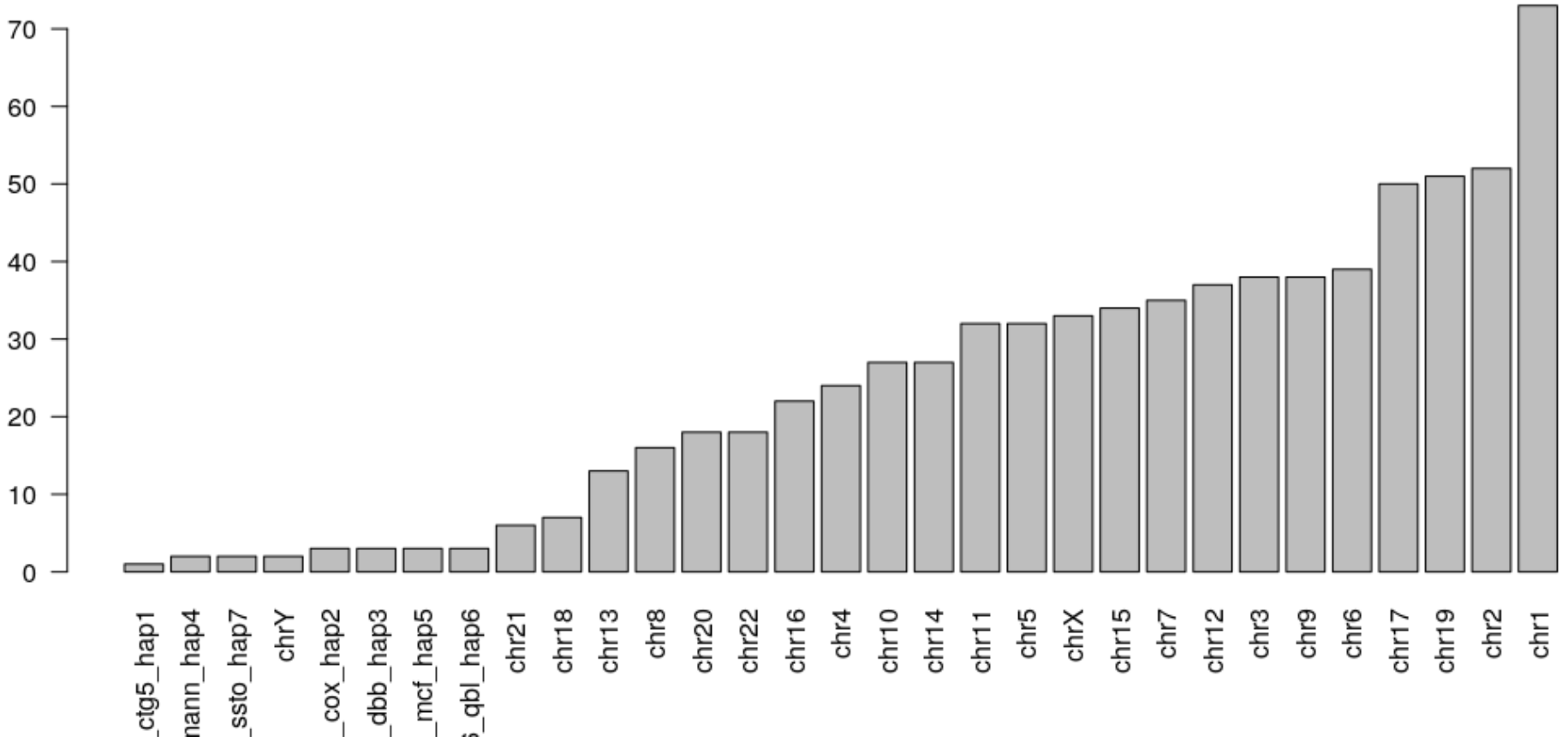
```
chr6 chr17 chr19  chr2  chr1
  39    50    51    52    73
```

# Kinases on chromosomes

5 lines of code …

```
> library(Homo.sapiens)
> genes <- select(Homo.sapiens, keys = keys(Homo.sapiens, keytype =
  "GENENAME"), columns = c("SYMBOL","TXCHROM"), keytype = "GENENAME")
> kinases <- genes[grep("kinase", genes$GENENAME), ]
> kinases_on_chr <- table(kinases$TXCHROM)
> barplot(kinases_on_chr[order(kinases_on_chr)], las = 2)
```

# Kinases on chromosomes



# help(rnorm)

Normal {stats}

## The Normal Distribution

### Description

Density, distribution function, quantile function and random generation for the normal distribution with mean equal to `mean` and standard deviation equal to `sd`.

### Usage

```
dnorm(x, mean = 0, sd = 1, log = FALSE)
pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
rnorm(n, mean = 0, sd = 1)
```

### Arguments

| | |
|---|---|
| `x,q` | vector of quantiles. |
| `p` | vector of probabilities. |
| `n` | number of observations. If `length(n) > 1`, the length is taken to be the number required. |
| `mean` | vector of means. |
| `sd` | vector of standard deviations. |
| `log, log.p` | logical; if TRUE, probabilities p are given as log(p). |
| `lower.tail` | logical; if TRUE (default), probabilities are $P[X \le x]$ otherwise, $P[X > x]$. |

### Details

If `mean` or `sd` are not specified they assume the default values of `0` and `1`, respectively.

The normal distribution has density

$$f(x) = 1/(\sqrt{(2\,\pi)}\,\sigma)\; e^{-((x-\mu)^2/(2\,\sigma^2))}$$

where $\mu$ is the mean of the distribution and $\sigma$ the standard deviation.

`qnorm` is based on Wichura's algorithm AS 241 which provides precise results up to about 16 digits.

### Value

`dnorm` gives the density, `pnorm` gives the distribution function, `qnorm` gives the quantile function, and `rnorm` generates random deviates.

### Source

For `pnorm`, based on

Cody, W. D. (1993) Algorithm 715: SPECFUN – A portable FORTRAN package of special function routines and test drivers. *ACM Transactions on Mathematical Software* **19**, 22–32.

For `qnorm`, the code is a C translation of

Wichura, M. J. (1988) Algorithm AS 241: The Percentage Points of the Normal Distribution. *Applied Statistics*, **37**, 477–484.

For `rnorm`, see RNG for how to select the algorithm and for references to the supplied methods.

### References


Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Johnson, N. L., Kotz, S. and Balakrishnan, N. (1995) *Continuous Univariate Distributions*, volume 1, chapter 13. Wiley, New York.


### See Also

Distributions for other standard distributions, including `dlnorm` for the *Log*normal distribution.

### Examples

```
require(graphics)

dnorm(0) == 1/ sqrt(2*pi)
dnorm(1) == exp(-1/2)/ sqrt(2*pi)
dnorm(1) == 1/ sqrt(2*pi*exp(1))
```

**http://www.bioinformin.net**