R workshop

Action!

(Operators and Functions)

Acting on variables

- In the previous section, we talked about objects and data sets

- Now let's do something with them
- Verbs
  - Operators
  - Functions

Operator

- Simple calculation

Operator

+	addition
-	subtraction
*	multiplication
/	division
^	taking powers

Order of Operations

- Important note: Order of operations matters
  - PEMDAS is your friend

```
> (8-4)/2
[1] 2
> 8-(4/2)
[1] 6
```

Example

- Take 14, add 4 and multiply the whole thing by 18.
- Take 14 and add the product of 4 and 18.

Example

- `> (14 + 4) * 18`
- `[1] 324`
- `> 14 + 4 * 18`
- `[1] 86`

Logical Operator

==	equality
!=	inequality
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

Logical Operator

- Returns a value of TRUE or FALSE

Example

- Test whether height is greater than 5.5

Logical Operators on Characters

- Height == 5.5
- Height = 5.5

- WARNING: a SINGLE equals sign will change your data!

```r
install.packages("babynames")
install.packages("ggplot2")

library(babynames)
library(ggplot2)

MyName <- "Sara"
birthday <- 1990
MySex <- "F"

data("babynames")
colnames(babynames)
myName.df <- subset(babynames, name == MyName)

ggplot(myName.df, aes(x = year, y = prop, color=sex)) +
        geom_line() +
        geom_point(aes(x = birthday,
                       y = myName.df[myName.df$name == MyName &
                           myName.df$year == birthday &
                           myName.df$sex == MySex, "prop"]),
                   color="black") +
        ggtitle(paste("Popularity of", MyName))
```

Functions

- Sometimes, you want to do more than add or multiply variables.

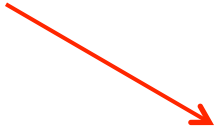
- To perform more complicated actions, use functions.
  - Functions are commands that describe, manipulate or analyze objects.

Functions

```
> log(10)
[1] 2.302585
```

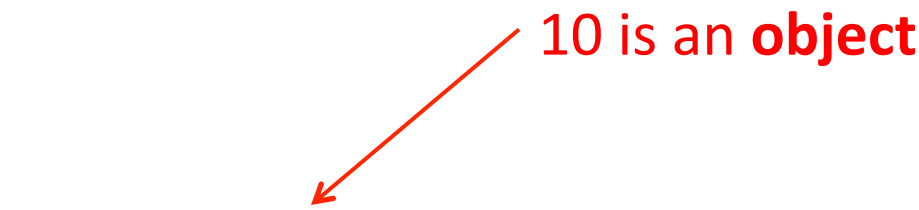
Functions

The **function** is called “log”.



```
> log(10)
[1] 2.302585
```

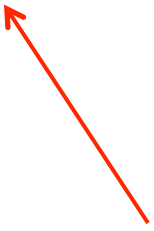
Functions




```
> log(10)
[1] 2.302585
```

Functions

```
> log(10)
[1] 2.302585
```



2.302…. is the **output** returned by the function

Mathematical functions

`sqrt()`	square root
`round()`	round a number
`log()`	logarithm
`exp()`	exponentiation
`abs()`	absolute value

Example

- Find the square root of 85.
- Take the log of 100.

Example

```
> sqrt(85)
[1] 9.219544
> log(100)
[1] 4.60517
```

```
install.packages("babynames")
install.packages("ggplot2")

library(babynames)
library(ggplot2)

MyName <- "Sara"
birthday <- 1990
MySex <- "F"

data("babynames")
colnames(babynames)
myName.df <- subset(babynames, name == MyName)

ggplot(myName.df, aes(x = year, y = prop, color=sex)) +
        geom_line() +
        geom_point(aes(x = birthday,
                       y = myName.df[myName.df$name == MyName &
                           myName.df$year == birthday &
                           myName.df$sex == MySex, "prop"]),
                   color="black") +
        ggtitle(paste("Popularity of", MyName))
```

Functions have three parts

- Function name
  - *Ex: log*
- Arguments
  - *Ex: 10*
- Output
  - *Ex: 2.302*

Functions have three parts

- Function name
  - *Ex: log*
- Arguments
  - *Ex: 10*
- Output
  - *Ex: 2.302*

Each function has one and only one name.

Functions have three parts

- Function name
  - *Ex: log*
- Arguments
  - *Ex: 10*
- Output
  - *Ex: 2.302*

One argument is always specified: the input. This is the object that the function acts on.

Other arguments control **how** the function acts. For example, do you want the natural log? Or log base 10?

Each function has defaults for its arguments. You should know what those are and how to change them.

Functions have three parts

- Function name
  - *Ex: log*
- Arguments
  - *Ex: 10*
- Output
  - *Ex: 2.302*

Output can be a:

- number/integer
- a TRUE/FALSE statement
- a character value
- all of the above

Output can be a:

- single value
- vector
- data frame
- matrix
- list

You can store the output by assigning it to another object.

An Analogy

- Catch the ball quickly

- Catch(item = ball, speed = “quickly”)

Exercise

- Use the seq() function to list numbers 0 to 100

- Arguments:
  - from: starting value of sequence
  - to: end value of sequence

Exercise

- Use the seq() function to list numbers 0 to 100, by intervals of 10

- Arguments:
  - from: starting value of sequence
  - to: end value of sequence
  - by: increment of the sequence

Exercise

- Use the seq() function to list numbers 0 to 100
- >`seq(from=0,to=100)`

- Now try to list numbers by intervals of 10
- >`seq(from=0,to=100,by=10)`

REMOVE POST-ITS!!

(and save!)

Multiple arguments

- Most functions take more than one argument.
- Separate arguments with commas.

Multiple arguments

- Most functions take more than one argument.
- Separate arguments with commas.

```
> round (x = 2.30467, digits = 3)
[1] 2.305
```

Multiple arguments

- Most functions take more than one argument.
- Separate arguments with commas.

```
> round (x = 2.30467, digits = 3)
[1] 2.305
```

Number that needs to be rounded.

Multiple arguments

- Most functions take more than one argument.
- Separate arguments with commas.

```
> round (x = 2.30467, digits = 3)
[1] 2.305
```

Number of digits to round to.

Arguments have Names

- Most arguments in functions have names.
- It is recommended that you use those names when using a function.

```
> round (x = 2.30467, digits = 3)
[1] 2.305
```

Order of arguments

- Technically, you don’t have to name the argument.
  - However, if you do not use the names, arguments MUST go in the right order.
  - Also you cannot skip arguments.
- If you name the arguments, you can put them in any order that you want and you can skip some.

Order of arguments

This is a bad idea!

- You don’t have to name the argument.
  - However, if you do not use the names, arguments MUST go in the right order.
  - Also you cannot skip arguments.
- If you name the arguments, you can put them in any order that you want and you can skip some.

Order matters!

```
> round (2.30467, 3)
[1] 2.305
> round (3, 2.30467)
[1] 3


> round (x = 2.30467, digits = 3)
[1] 2.305
> round (digits = 3, x = 2.30467)
[1] 2.305
```

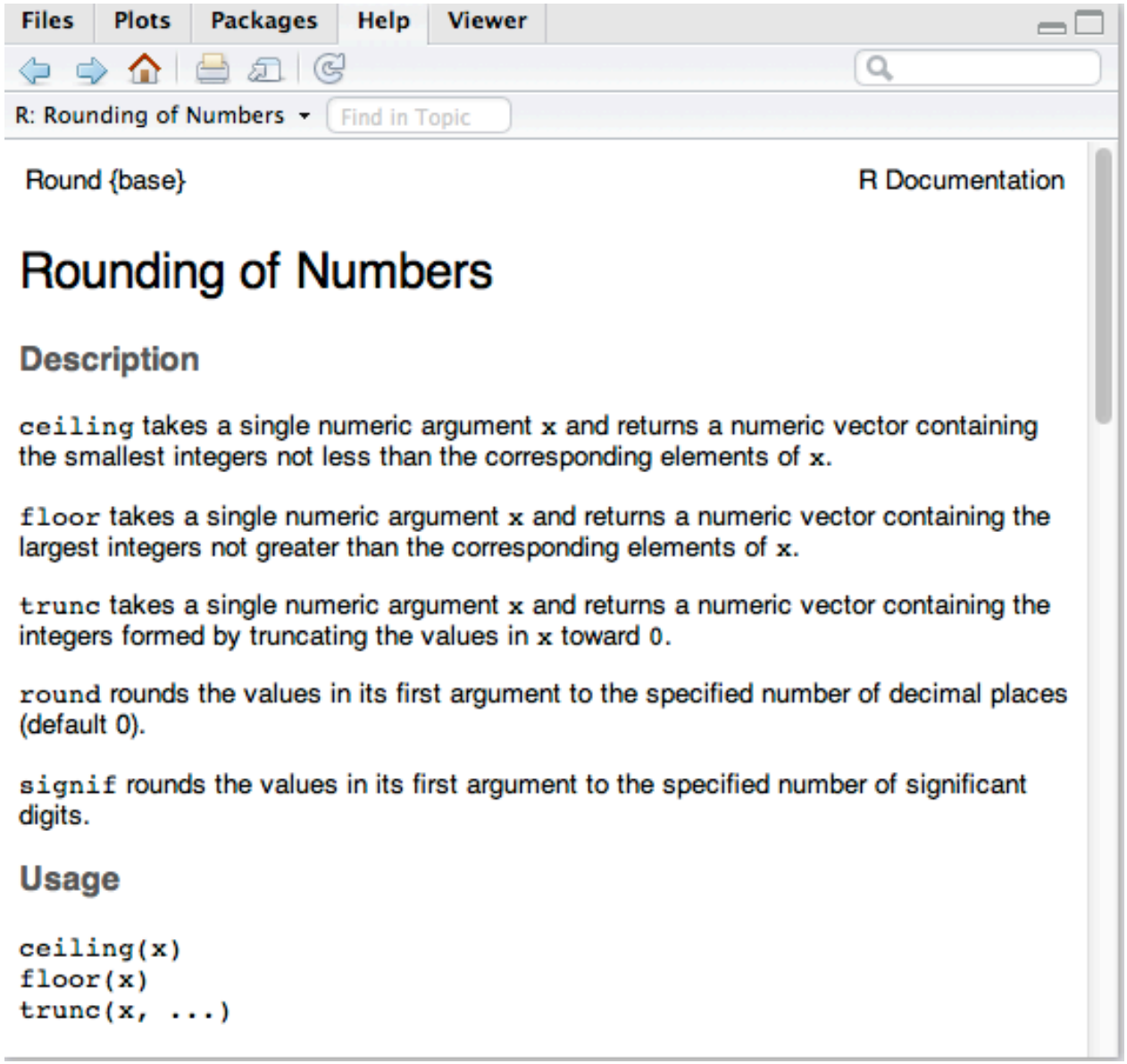
```
install.packages("babynames")
install.packages("ggplot2")

library(babynames)
library(ggplot2)

MyName <- "Sara"
birthday <- 1990
MySex <- "F"

data("babynames")
colnames(babynames)
myName.df <- subset(babynames, name == MyName)

ggplot(myName.df, aes(x = year, y = prop, color=sex)) +
        geom_line() +
        geom_point(aes(x = birthday,
                       y = myName.df[myName.df$name == MyName &
                           myName.df$year == birthday &
                           myName.df$sex == MySex, "prop"]),
                   color="black") +
        ggtitle(paste("Popularity of", MyName))
```

Great, but how do I know what the arguments are?

- Look in the R documentation

```
> ?round
```

- In RStudio, hit tab to see names of arguments and descriptions.


Files
Plots
Packages
Help
Viewer
R: Rounding of Numbers
Find in Topic
Round {base}
R Documentation
Rounding of Numbers
Description
ceiling takes a single numeric argument x and returns a numeric vector containing the smallest integers not less than the corresponding elements of x.
floor takes a single numeric argument x and returns a numeric vector containing the largest integers not greater than the corresponding elements of x.
trunc takes a single numeric argument x and returns a numeric vector containing the integers formed by truncating the values in x toward 0.
round rounds the values in its first argument to the specified number of decimal places (default 0).
signif rounds the values in its first argument to the specified number of significant digits.
Usage
ceiling(x)
floor(x)
trunc(x, ...)

You try!

- Look up documentation for the correlation function, cor()

You try!

```
> ?cor
```

R: Correlation, Variance and Covariance (Matrices)

cor {stats} R Documentation

Correlation, Variance and Covariance (Matrices)

Description

`var`, `cov` and `cor` compute the variance of `x` and the covariance or correlation of `x` and `y` if these are vectors. If `x` and `y` are matrices then the covariances (or correlations) between the columns of `x` and the columns of `y` are computed.

`cov2cor` scales a covariance matrix into the corresponding correlation matrix *efficiently*.

Usage

```
var(x, y = NULL, na.rm = FALSE, use)

cov(x, y = NULL, use = "everything",
    method = c("pearson", "kendall", "spearman"))

cor(x, y = NULL, use = "everything",
    method = c("pearson", "kendall", "spearman"))

cov2cor(V)
```

Arguments

`x`	a numeric vector, matrix or data frame.
`y`	`NULL` (default) or a vector, matrix or data frame with compatible dimensions to `x`. The default is equivalent to `y = x` (but more efficient).
`na.rm`	logical. Should missing values be removed?
`use`	an optional character string giving a method for computing covariances in the presence of missing values. This must be (an abbreviation of) one of the strings `"everything"`, `"all.obs"`, `"complete.obs"`, `"na.or.complete"`, or

cor {stats}

Correlation, Variance and Covariance (Matrices)

cor {stats}

Correlation, Variance and Covariance (Matrices)

Description

`var`, `cov` and `cor` compute the variance of `x` and the covariance or correlation of `x` and `y` if these are vectors. If `x` and `y` are matrices then the covariances (or correlations) between the columns of `x` and the columns of `y` are computed.

`cov2cor` scales a covariance matrix into the corresponding correlation matrix *efficiently*.

cor {stats}

Correlation, Variance and Covariance (Matrices)

Description

`var`, `cov` and `cor` compute the variance of `x` and the covariance or correlation of `x` and `y` if these are vectors. If `x` and `y` are matrices then the covariances (or correlations) between the columns of `x` and the columns of `y` are computed.

`cov2cor` scales a covariance matrix into the corresponding correlation matrix *efficiently*.

Usage

```
var(x, y = NULL, na.rm = FALSE, use)

cov(x, y = NULL, use = "everything",
    method = c("pearson", "kendall", "spearman"))

cor(x, y = NULL, use = "everything",
    method = c("pearson", "kendall", "spearman"))

cov2cor(V)
```

cor {stats}

Correlation, Variance and Covariance (Matrices)

Description

`var`, `cov` and `cor` compute the variance of `x` and the covariance or correlation of `x` and `y` if these are vectors. If `x` and `y` are matrices then the covariances (or correlations) between the columns of `x` and the columns of `y` are computed.

`cov2cor` scales a covariance matrix into the corresponding correlation matrix *efficiently*.

Usage

```
var(x, y = NULL, na.rm = FALSE, use)

cov(x, y = NULL, use = "everything",
    method = c("pearson", "kendall", "spearman"))

cor(x, y = NULL, use = "everything",
    method = c("pearson", "kendall", "spearman"))

cov2cor(V)
```

Arguments

`x`	a numeric vector, matrix or data frame.
`y`	`NULL` (default) or a vector, matrix or data frame with compatible dimensions to `x`. The default is equivalent to `y = x` (but more efficient).
`na.rm`	logical. Should missing values be removed?
`use`	an optional character string giving a method for computing covariances in the presence of missing values. This must be (an abbreviation of) one of the strings `"everything"`, `"all.obs"`, `"complete.obs"`, `"na.or.complete"`, or `"pairwise.complete.obs"`.
`method`	a character string indicating which correlation coefficient (or covariance) is to be computed. One of `"pearson"` (default), `"kendall"`, or `"spearman"`, can be abbreviated.
`V`	symmetric numeric matrix, usually positive definite such as a covariance matrix.

Details

For `cov` and `cor` one must *either* give a matrix or data frame for `x` *or* give both `x` and `y`.

The inputs must be numeric (as determined by `is.numeric`: logical values are also allowed for historical compatibility): the `"kendall"` and `"spearman"` methods make sense for ordered inputs but `xtfrm` can be used to find a suitable prior transformation to numbers.

Details

For `cov` and `cor` one must *either* give a matrix or data frame for `x` *or* give both `x` and `y`.

The inputs must be numeric (as determined by `is.numeric`: logical values are also allowed for historical compatibility): the `"kendall"` and `"spearman"` methods make sense for ordered inputs but `xtfrm` can be used to find a suitable prior transformation to numbers.

Value

For `r <- cor(*, use = "all.obs")`, it is now guaranteed that `all(r <= 1)`.

Details

For `cov` and `cor` one must *either* give a matrix or data frame for `x` *or* give both `x` and `y`.

The inputs must be numeric (as determined by `is.numeric`: logical values are also allowed for historical compatibility): the `"kendall"` and `"spearman"` methods make sense for ordered inputs but `xtfrm` can be used to find a suitable prior transformation to numbers.

Value

For `r <- cor(*, use = "all.obs")`, it is now guaranteed that `all(r <= 1)`.

Examples

```
var(1:10)  # 9.166667

var(1:5, 1:5) # 2.5

## Two simple vectors
cor(1:10, 2:11) # == 1

## Correlation Matrix of Multivariate sample:
(Cl <- cor(longley))
## Graphical Correlation Matrix:
symnum(Cl) # highly correlated
```

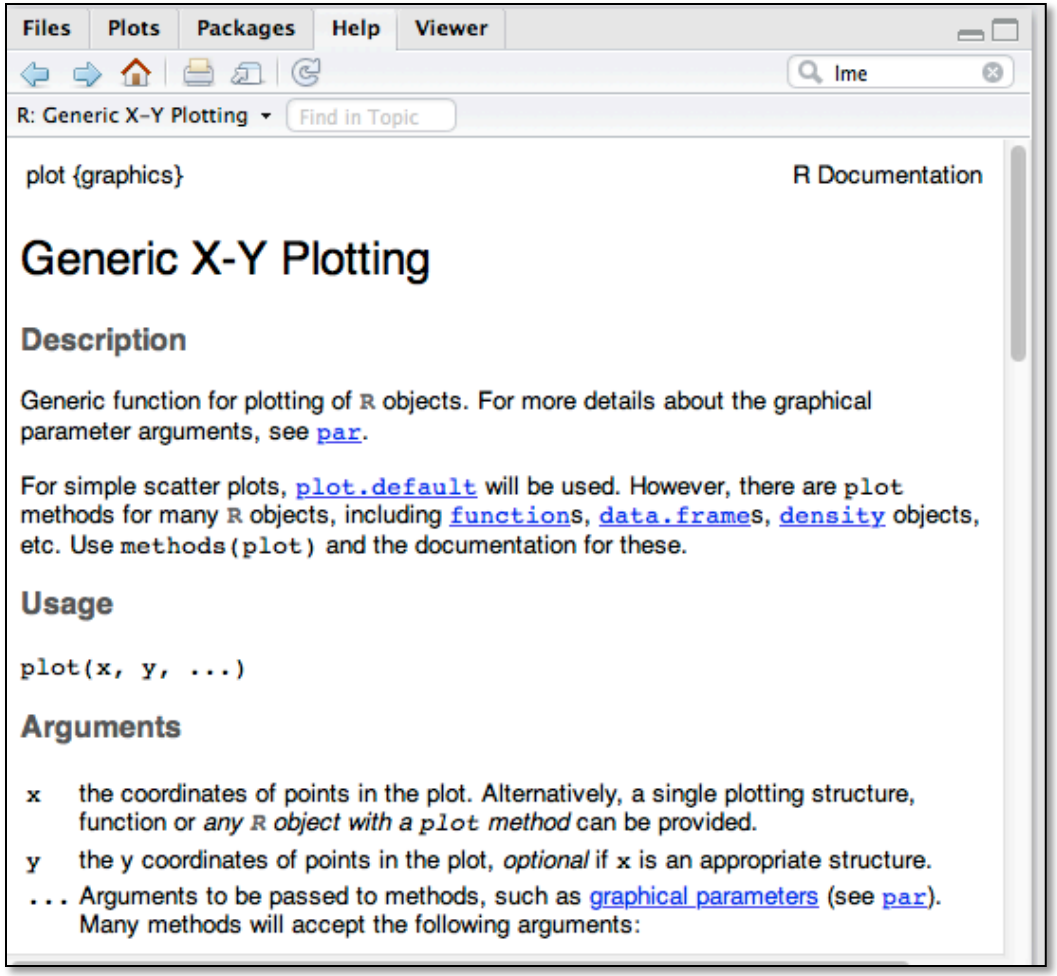
Example

- 5 volunteers, how many cups of coffee did you drink today?

- Create a vector, add to data frame

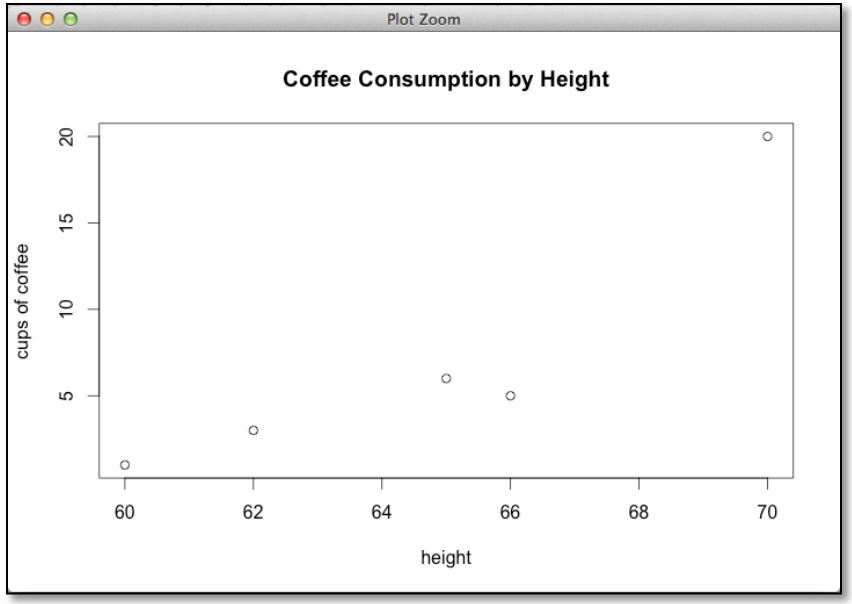
Example

- Look up documentation for plot
- Make a scatter plot of coffee by height
- Add a title to your graph
- Add labels to x and y axes
- What happens if you add type='l'
  - What is the default for type?


Files
Plots
Packages
Help
Viewer
lme
R: Generic X-Y Plotting
Find in Topic
plot {graphics}
R Documentation
Generic X-Y Plotting
Description
Generic function for plotting of R objects. For more details about the graphical parameter arguments, see par.
For simple scatter plots, plot.default will be used. However, there are plot methods for many R objects, including functions, data.frames, density objects, etc. Use methods(plot) and the documentation for these.
Usage
plot(x, y, ...)
Arguments
x the coordinates of points in the plot. Alternatively, a single plotting structure, function or any R object with a plot method can be provided.
y the y coordinates of points in the plot, optional if x is an appropriate structure.
... Arguments to be passed to methods, such as graphical parameters (see par). Many methods will accept the following arguments:

R Code

```
>plot(x=classdata$height,
y=classdata$coffee,
main="Coffee Consumption by Height”,
xlab="height (inches)",
ylab="cups of coffee")
```



Back to the Documentation!

Arguments

`x` the coordinates of points in the plot. Alternatively, a single plotting structure, function or *any* `R` *object with a* `plot` *method* can be provided.

`y` the y coordinates of points in the plot, *optional* if `x` is an appropriate structure.

`...` Arguments to be passed to methods, such as graphical parameters (see `par`). Many methods will accept the following arguments:

`type`

what type of plot should be drawn. Possible types are

- `"p"` for **p**oints,
- `"l"` for **l**ines,
- `"b"` for **b**oth,
- `"c"` for the lines part alone of `"b"`,
- `"o"` for both '**o**verplotted',
- `"h"` for '**h**istogram' like (or 'high-density') vertical lines,
- `"s"` for stair **s**teps,
- `"S"` for other **s**teps, see 'Details' below,
- `"n"` for no plotting.

All other `type`s give a warning or an error; using, e.g., `type = "punkte"` being equivalent to `type = "p"` for S compatibility. Note that some methods, e.g. `plot.factor`, do not accept this.