|  |  |
| --- | --- |
| **R-Konventionen** |  |
| Case-Sensitivity =  Berücksichtigung von Groß- und Kleinschreibung | * R differenziert stets zwischen Groß- und Kleinschreibung * im Englischen sagt man auch: **R ist ‚case-sensitive‘** |
| Kommentare einfügen | > # <Kommentar>   * gehen immer bis zum Zeilenende * können mit 2 Ausnahmen überall eingefügt werden: nicht innerhalb von einer Zeichenkette, und nicht innerhalb einer Funktion |
| Befehle (commands) | * Mehrere Befehle können entweder durch ;  oder durch das Nutzen der nächsten Zeile voneinander getrennt werden * [> sink(‚name of file‘): will record every following output to an external file 🡪 note: output is then not displayed in console anymore!] |
| Zahlen (numbers) & spezielle Werte | * Dezimaltrenner ist ein **Punkt** * Zahlen können im Exponentialformat eingegeben werden   + Das Format Exponentialzahl zeigt eine Zahl in exponentieller Notation an, wobei ein Teil der Zahl durch "En" ersetzt wird. Dabei multipliziert "E" (steht für Exponent) die vorangehende Nummer mit 10 zur Potenz n   + Ein wissenschaftliches Format mit zwei Dezimalstellen zeigt 12345678901 als 1,23E+10 an, das bedeutet, 1,23 mal 10 zur Potenz 10   + in R lautet die Schreibweise wie folgt (z.B.): 3.4E5 * Spezielle Werte: * Unendlich: Inf / - Inf   + z.B. 1/0 ergibt Inf   + obwohl es mathematisch nicht definiert ist, die Division eines Wertes ≠ 0 mit 0 wird immer Inf ergeben * **Boolesche Variablen (Boolean Variables):** TRUE (or T) and FALSE (F)   + if one uses Boolean Variables in functions, T and F are treated as follows: **T = 1 and F = 0** (see datatype “logical”)   + for example: * **Nicht verfügbar: NA (Not available); missing values, or better: not assigned value**   + for more information: R language definitions, NA handling   + a missing value is a value which is unknown, hence R treats it as a value that is not assigned   + think of NA as meaning "I don't know what's there" 🡪 also mehr als eine Art “Platzhalter”   + NA is not a string nor a numeric value   + the **default data type** of **NA is logical**, unless it gets coerced to some other type   + numeric and logical calculations with NA generally return NA   + however, in cases where the result of the operation would be the same for all possible values the NA could take, the operation may return this value     - e.g. NA^0 = 1     - also, in particular: see cases chapter 5, section 3.2.5 Filter out NA * NULL is a special object   + it is used whenever there is a need to indicate or specify that an object is absent   + It should not be confused with a vector or list of zero length   + NULL has no datatype and no modifiable properties   + you cannot set attributes on NULL 🡪 overall, the difference between NULL and NA is not very consistent * Math. Error: NaN (Not a Number)   + z.B. 1/0 – 1/0 ergibt NaN   + This exists only in the double data type   Pi: pi |
| Checking for special values | Syntax: is.<special value>(<variable>)   * + **returns logical vector** * NA values: is.na(<variable>) * NaN: is.nan(<variable>) * Null: is.null(<variable>) |
| Objekte (objects) =  data structures (high level view) | Best summary (Ren, p. 73): In fact, in R environment, **everything we use is an object**, **everything we do is a function,** and, maybe to your surprise, **all functions are still objects**  Cran R language definitions, p. 7:   * in every computer language, variables provide a means of accessing the data stored in memory * R does not provide direct access to the computer’s memory but rather provides a number of **specialized data structures**, which are called **objects** * these objects are referred to through symbols or variables * in R, however, these symbols are themselves objects and can be manipulated in the same way as any other object  🡪 this is different from any other language and has wide ranging effects * the **entities R creates and manipulates, i.e. operates on,** are **technically known as objects** (Objekte sind also in R instanziiert und durch R interpretiertes) * Es gibt **verschiedene Sorten von Objekten (=Datenstrukturen) in R, in denen Daten gespeichert werden können:** variables, arraysof numbers, character strings, expressions, functions or more general structures build from such components, such as:   + Vektoren (vectors): collection of objects of the same atomic datatype   + Matrizen (matrices) (n x m): collection of objects of the same atomic datatype   + Arrays: collection of matrices   + Listen (lists): collection of objects of potentially different datatypes   + Data frames / Tibbles: (n x m) ‘matrix‘ where each column can have a different datatype * every task involves different types of objects and each object has a different goal and behavior |
| Check and coerce to specific object type | 🡪 I already indicated it with special values; it is always:  > is.<Type of Object>(<Object>) 🡪 checks whether object is of a specific type 🡪 returns logical vector  🡪 e.g.:      > as.<Type of Object>(<Object>) 🡪 **coerces** object into another type  🡪 e.g.:     * > showMethods(coerce)**shows all the possible conversions from one object to another** |
| Data types (low level view) | * **Basic (atomic) data types:**    + **logical (lgl):** Boolean Variable, hence TRUE (or T), FALSE (or F)     - **Logical vectors are coerced to integer vectors in all contexts where a numerical value is required**, with ‘TRUE’ being mapped to ‘1L’, ‘FALSE’ to ‘0L’ and ‘NA’ to ‘NA\_integer\_’   + **character (chr):** **character strings** / character vectors     - Eine **Zeichenkette** oder (aus dem Englischen) ein **Character String** ist in der [Informatik](https://de.wikipedia.org/wiki/Informatik) eine Folge von [Zeichen](https://de.wikipedia.org/wiki/Zeichen) (z. B. Buchstaben, Ziffern, [Sonderzeichen](https://de.wikipedia.org/wiki/Sonderzeichen))     - A character string is a series of characters manipulated as a group     - A character string differs from a name in that it does not represent anything - a name stands for some other object     - Werden zu Beginn und zu Ende durch >“ oder > ‘ begrenzt   z.B.: > “ABC” oder > ‘abc’   * + - **beachte**: das Apostroph ist nur bei entsprechendem Kontext notwendig, d.h. wenn Charakter-String auch als Buchstaben-Kette, d.h. als Text, interpretiert werden soll (z.B. bei der Namensgebung)  und nicht bspw. als Variable definiert wird und damit **eig. einen anderen atomic data type hat**   + **double (dbl):** floating point numeric numbers [Gleitkommazahlen; Gleitkommazahlen sind der heute übliche Weg, um Kommazahlen, also ***reelle Zahlen***, für den Computer darzustellen; Gleitkommazahlen sind nur Nährungen; denn z.B. die Division von 10 durch 3 würde zu unendlich vielen Nachkommastellen führen, die durch den Computer aufgrund von beschränkter Rechen- und Speicherkap. kaum darstellbar ist]   + **integer (int):** integer values, i.e. values from Bildergebnis für menge z     - integers in R have to be written with an L following the number     - the L tells R to store the number as an integer   + **date + time (ddtm)**   + **dates (date)**   + **factors (fctr)**: R uses factors to represent categorical variables with fixed possible values   + **complex:** as in complex numbers; write as 0+0i * R ist eine interpretierte Programmiersprache: sie will es dem Anwender ermöglichen Definitionen und Konkretisierungen flexibel zu handhaben * aus Geschwindigkeitsgründen versucht R Auswertungen so spät wie möglich durchzuführen * dies erfordert einige Einschränkungen an die Sprache, die R von anderen Programmiersprachen unterscheidet * **so kennt R keine abstrakten Datentypen: ein Datentyp ist durch seine Instanzen, die Variablen definiert**    + ein abstrakter Datentyp besteht aus einem Wertebereich (d.h. einer Menge von Objekten) und darauf definierten Operationen * **der Datentyp einer Variablen ist dynamisch**: derselbe Name einer Variablen in demselben Kontext kann ***zu unterschiedlichen Zeiten unterschiedliche Variablenwerte und -typen kennzeichnen*** * zu einem gegebenen Zeitpunkt hat die Variable jedoch stets einen bestimmten Typ * das R-Typensystem versteht man am Besten, wenn man sich seine Historie anschaut:   + zunächst wurde der Typ beschrieben durch: > mode(<name of object>)     - aus Cran R-language definition: ‘Function mode gives information about the mode of an object in the sense of Becker, Chambers & Wilks (1988), and is more compatible with other implementations of the S language’   + diese Funktion wurde jedoch durch > typeof(<name of object>) weitgehend abgelöst     - type-Funktion beschreibt also was der Datentyp eines Objektes ist   + die folgende Übersicht beschreibt – ausschöpfend, d.h. über die Basistypen hinausgehend - die Datentypen, die durch die typeof-Funktion derzeit berichtet werden:      * + **komplexere Datentypen werden auf die in der Übersicht definierten zurückgeführt, indem die Variablen mit Attributen versehen werden:**     - so sind Matrixen oder Arrays nur spezielle Vektoren, die sich dadurch auszeichnen, dass sie ein dim-Attribut haben * **Datentypen in einem data frame:**    + in order to check the datatypes in a data frame, one may simply run the dataframe   + **the row right above the values always states the data type of the respective variable in short** |
| Checking for and coercing to **a specific** datatype | * same principle as for checking and coercing objects * **Checking:** is.<datatype>(<variable>) * **Coercing:** as.<datatype>(<variable>)      * **note as.numeric:**    + “numeric” is the data mode lying behind the data type within which R stores the data; “integer” and “double” are both “numeric”   + R will automatically switch between those two datatypes if necessary   + if not explicitly specified as integer, integer values will be typed as “double” as a lot of computational processes require this * again > showMethods(coerce)**shows all the possible conversions from one datatype to another** |
| More about an object: structure and class of an object (high-level view) | * die Funktion > str(<name of object>) liefert neben dem **Datentyp** eines Objektes zusätzlich noch die einzelnen Objektelemente: * > class(‘name of object’) * is used to define/identify what "type" an object is **(high-level view)** from the point of view of object-oriented programming in R * note: object-oriented programming in R is really good for building tools that can be used in data analysis, but bad for data analysis itself; for the latter functional programming is used   + w/ functional programming you think about the functions first and then about the data structure   + w/ object-oriented programming you think about the data structures first   + So for…   > x <- 1:3  > class(x)  [1] "integer"   * + …any generic function (in object-oriented programming it is differentiated between ‘generic’ (e.g. print function) and ‘method’ function) that has an "integer" method will be used * Alternative, um nach dem Objekttyp zu prüfen:  > is.‘zu erfüllende Eigenschaft‘(‚name des Objektes‘) e.g. is.integer(<variable>) * **class vs. mode vs. typeof** |
| Länge eines Objektes | * length of an object: e.g. length of a vector **:**   > length(‘name of object’) |
| Attributes of an object | * **attributes are basically metadata of an object** * all objects, except NULL, can have one or more attributes attached to them * attributes can be thought of as a named list: they can be of any datatype and therefore include any kind of **information/metadata about the object**   + the attribute dimnames, for instance, is a list containing the row and column names of a matrix * some attributes are proprietary to R   + they form one of the mechanisms R uses to define specific object types   + e.g. matrices and arrays are simply vectors with the attributes dim (and optionally dimnames) attached to the vectors * yet, all objects can have **arbitrary [beliebig] additional attributes**, used to **store metadata** about the object 🡪 attributes are really great for storing contextual metadata about e.g. a variable * one of the most common are:   + names for all sorts of objects   + dimnames for matrices   + class of an object   + comments for any R objects * Excursus: comment function comment()   + contrary to other attributes, this function is not printed   + the comment function simply stores the string as an attribute of an object   + it can be very useful to add some further explanations to a variable, if you want to save the variables besides your source code 🡪 if your model takes more than ten minutes to run the code, it might be useful to save the source code as well as the variables   + set comment: > comment(<Object>) <- “Comment”   + query comment: > comment(<Object>)   + for example: * set individual attributes with:  attr(<object>, “<name of the attribute>”) <- “<the information/metadata to store>” * note: one can either use ‘ or “ apostrophe * for example:      * query attribute: attr(<name of object>,”<name of the attribute>”)      * + if the attribute I am asking for doesn’t exist, R simply returns NULL * To see all attributes of an object at once (which is probably easier, especially if you’re getting back to a project after a couple of months and actually do not remember the attributes by name) you use:  attributes (<name of object>)  🡪 R then returns a named list where each item in the list is an attribute * for example:   🡪   * **Caution:** by default, most attributes are lost, when an object is modified * the only attributes not lost are the three most important:   + Names   + Dimensions   + Class * **remove attributes:** > attributes(<name of object>) <- NULL |
| Naming objects (= Festlegen von Variablen) | * **Names can be seen as build-in attributes** (Rintro Part 1, slide 28) * **Zugelassene Symbole:** * alle alpha-numerischen Symbole (ausgenommen Umlaute) * sowie . und \_ * R-Syntax:   + ein Name muss mit einem Buchstaben oder . beginnen   + beginnt er mit . darf das zweite Zeichen (logischerweise) keine Ziffer sein * Namen können unendlich viele Zeichen haben * Konvention für Namen mit mehreren Wörtern: wort1\_wort2\_wort3 * guter Stil ist außerdem alles klein zu schreiben, um Gefahr von Inkonsistenzen zu mindern * Bereits belegte Namen sind z.B.: pi, t, f, T, F, mean, var, ... (können aber überschrieben werden) * **Zuweisung von Namen: die Zuweisung eines Namens zu einem Objekt, erlaubt es auf das jew. Objekt anhand des Namens zuzugreifen** * > <Name> <- Objekt * Beispiele:   + > x <- 3   + > x <- c(2, 3.5, 4) * **beachte:** es ist möglich in einem Command mit einem zuvor definiertem Objekt zu arbeiten, und diesem im gleichen command wiederrum den gleichen Namen zu geben 🡪 Inhalt wird dann einfach überschrieben * z.B.: * To display all names of currently stored objects: > objects()   + note: **the collection of objects currently stored is called *workspace***   + thus, the list of all stored objects **can also be found in the environment pane** * to display all names assigned to an object: names(<object>) * **Ändern und Zuweisung von Namen mit names-Function** > names(): * **1) Ändern/Zuweisung aller Namen von Elementen eines Vektors:** > names(<name of vector>) <-c(‚new name of element 1‘, …, ‚new name of element n‘)   + for example: 🡪   + 🡪 beachte: sollen hier bereits vergebene Namen beibehalten werden, müssen diese an entsprechender Stelle ihres Indexes genannt werden 🡪 R geht hier nach Position von links nach rechts vor * **2) Ändern des Namens eines Elementes eines Vektors:** > names(<name of object>)[names(<name of object>)==’current name of element’]<-‘new name of element’ 🡪 erst wird auf den Namen des Objektes zugegriffen 🡪 dann wird die Namensliste beschränkt (== meint exactly equal to) * **3) Ändern eines Teils des Namens eines Elementes eines Vektors**:  names & gsub Funktion  > names(<name of object>) <- gsub(‘<part that should be renamed>’, ‘<new part>’, names(<name of object>))   + for example:   🡪  **There is a much easier shortcut to do all of this:**   * go into console * type the name or the first elements of the name * click: STRG + ↑ * then a list of all commands one has typed so far will appear * choose the one you need * change what you need 🡪 will add this as new object * press ENTER * delete object w/ old name from workspace in Global environment * **Zuweisung von Namen mit attributes-Function:** * **> attr(<name of vector>, “names“) = c(”<names of objects>”)** * for example: * to remove names: NULL   + for example:      * + **careful: dann werden auch die ursprünglichen Namen gelöscht** |
| **Kategoriale und Kontinuierliche Variablen** | * **Kategoriale Variablen:**   + sind Merkmale, die eine begrenzte Anzahl an Ausprägungen haben   + z.B.: Geschlecht, Religion, Parteipräferenz * **Kontinuierliche Variablen:**    + Variablen mit sehr vielen Ausprägungen, werden nicht zu den kategorialen Variablen; liegt diesen Messungen eine kontinuierliche Eigenschaft zugrunde, werden sie als kontinuierliche Variable bezeichnet * kategoriale vs. kontinuierliche Variablen, sind nicht zu verwechseln mit diskreten und stetigen Zufallsvariablen |
| **Functions** | * **functions are objects that form part of the R system along w/ similar user written functions** * eine Funktion in R hat immer die folgende Form und wird entsprechend aufgerufen:  > <Name der Funktion>(<Parameter>)   + z.B.: > e\_10 = exp(10) * **eine Funktion muss allerdings keine Argumente besitzen** * z.B.: >q() 🡪 dies ist die sog. quit-Prozedur, die aufgerufen werden kann um R zu beenden |
| Expressions | * Expressions as objects, form an advanced part of R |
| **Operatoren (operators)** | * <-/=: Zuweisungen (z.B. von Namen)   + beachte: an mehreren Stellen wird von der **Nutzung des =-Zeichen abgeraten**! * **Shortcut für Erstellung des <- Zeichens in R: ALT+-** |
| Subsetting & extracting operators (rectangular brackets and $) | * []: **single object** can be selected using **integer indices or character strings**   + used for sub-setting:     - creating a new vector out of given one     - creating a new list out of given one   + note: [] “peels off” one layer of the respective objective     - applied to vectors: we get directly to values or named values     - applied to matrices: we get directly to values 🡪 if we subset one-dimensionally we get vector as object type 🡪 if we subset two-dimensionally we get matrix as object type     - applied to lists: we get to list element (which may be a vector, matrix or other list) 🡪 we get list as object type     - applied to data frames: we get column(s) note when choosing only one column:       * if we subset data frame ‘as’ list we still get a data frame       * if we subset data frame ‘as’ matrix we get vector * [[]]: extract **single elements/components** from a vector, matrix, array or list by using **integer indices or character strings**; **names are dropped** * $: **single elements/components** from a matrix, list or data frame can be extracted by character strings/names; **names are dropped** * [[]] & $ **peel off 2 layers** |
| Arithmetische Operatoren (arithmetic operators) | * +, -, \*, /   + note: **R berücksichtigt Punkt-vor-Strich** * \*\* or ^: exponentiation * (special case %\*%: multiplication of matrices) |
| Modular arithmetisch | * Modulare Arithmetik ist ein Teilbereich der Arithmetik, bei dem besonderes Interesse am Rechnen vorhanden ist * <number>**%/%**<number>: **ganzzahlige Division**; z.B. 16%/%5=3 * <number>**%%**<number>: **Rest**; z.B.: 16%%5 = 1 |
| Relationale Operatoren (relational operators) | * **use** > **?Comparison for help on those operators** * note: if those operators are used to **compare** two objects the **result** will (at first) always be **boolean variables** TRUE/FALSE * <, >, <=, >= * == : (exactly) equal to * near() function: safe way of comparing if two vectors of **floating** point numbers are (pairwise) equal. This is safer than using ==, because it has a built in tolerance   + e.g.: * != : not equal to * %in% : compares whether every element on left-hand side vector is included in the right-hand side vector   + for example      * **diff. between == and %in%:**    + bei == spielt die Reihenfolge eine Rolle; R vergleicht hier zeilenweise, ob Werte übereinstimmen   + bei %in% wird überprüft, ob der erste/zweite etc. Wert des linken Vektors an irgendeiner Stelle im rechten Vektor auftaucht |
| Logische Operatoren (logical operators) = Boolsche Operatoren | * ebenso wie bei relationalen Operatoren, ist auch bei logischen Operatoren das Resultat eine boolesche variable, wenn die Operatoren zum Vergleich von Objekten genutzt werden   + dann prüft R grundsätzlich, ob ein Wert bzw. ein Ausdruck Wahr ist * sofern 2 oder mehrere Vektoren über logische Operatoren miteinander verknüpft werden, überprüft R auf Werte = 0 (⬄ TRUE) bzw. ≠ 0 (⬄ FALSE)   + keep in mind **in that context**: **TRUE corresponds to non-zero scalar** and **zero scalar is considered as FALSE (see datatype “logical”)** * sofern zwei oder mehrere Ausdrücke (z.B. x > 2 & x < 5) über logische Operatoren miteinander verknüpft werden, überprüft R, ob diese Ausdrücke an sich Wahr oder Falsch sind * Examples of using logical operators with vectors: * |: Element-wise OR   + for example  results in:  as R tests whether either one of the vectors contains non-zero values * ||: Logical OR; R examines only the first element of the objects   + for example results in * &: Element-wise logical AND   + for example results in  as R tests whether both vectors contain non-zero values * &&: Logical AND; R examines only the first element of the objects   + for example results in * xor(x, y): exclusive OR (entweder oder)   + for example  results in  as R tests whether only one vector at a time contains non-zero values * !: Logical NOT 🡪 Komplementärereignis e.g. P(A) = (1-P(A))   + kehrt den Befehl also um   + for example results in   + !x fragt danach, ob der Vektor x **keine** Werte ≠ 0, also Werte = 0, beinhaltet 🡪 TRUE korrespondiert mit Werten = 0  🡪 FALSE korrespondiert mit Werten ≠ 0 * any (): **Given a set of logical vectors**, are any of the values true?   + for example: results in   + but results in * all (): **Given a set of logical vectors**, are all of the values true |
| Logarithmus & Exponentialfunktion | * exp(<number>): z.B. exp(3) entspricht e^3 mit e= eulerische Zahl * log(<number>): natürlicher Logarithmus; z.B.: log(10) entspricht ln(10) * log<Basis>(<number>): z.B: log2(10) entspricht log. von 10 zur Basis 2   + generally, log2 () is recommended   + this is due to     - a doubling on a normal scale, corresponds w/ + 1 on log2 scale     - and halving on a normal scale, corresponds w/ -1 on log2 scale     - e.g.: log2(8) = y ⬄ 2y = 8 ⬄ y = 3 log2(16) = y ⬄ 2y = 16 ⬄ y = 4 |
| Runden | * round(<variable>, n): round to n decimal places (kaufm. runden) * floor(): **abrunden** auf nächste ganze Zahl * ceiling(): **aufrunden** auf nächste ganze Zahl * trunc(): abschneiden der Nachkommastellen |
| Weitere Operatoren | * sqrt(): Quadratwurzel * abs(): Betrag * choose(<n>,<r>) with n: n elements, and r: r subset elements   + Kombination ohne Wiederholung: Ziehen ohne Zurücklegen; Reihenfolge egal * factorial(<n>): n-Fakultät * rank(<variable>: Rank of elements (more details on rank-functions in Chapter 5, dplyr) |
| Aggregational functions | * Aggregate functions perform a calculation on a set of values and return a single value * sum(<numbers>): Summe * prod(<numbers>): Produkt * diff(<numbers>): Differenz * mean(<numbers>): Durchschnitt * median(<numbers>): Median * max(<numbers>) * min(<numbers>) * cor(<var1>, <var2>): Correlation * var(<var>): Varianz * sd(<var>): Standardabweichung * quantile(<var>): Quantile |
| Trigonometrische Funktionen | * sin(), cos(), tan() * inverse trigonometrische Funktionen: asin(), acos(), atan() |