DATA TYPES AND DATA STRUCTURES

mgr Maria Kubara, WNE UW

REMINDER

Variable – used for storing data and results of the operations.

Function - sequence of instructions, making up for a block of code, which can be used multiple times in different places. Functions usually return certain values after finishing their work.

```
# commenting with a hash symbol

function(argument1, argument2, argument3, ...)

a <- "assigning value to a variable"

b = "alternative way of assigning values"

c <- 2

d = 3

help(functionName) #finding documentation for a function

?functionName #finding documentation for a function
```

BASIC DATATYPES IN R

DATA TYPES

Data, which we are saving and storing in R can have different types. Depending on the type of the data, different functions and operations will be accessible.

There are 5 main datatypes in R:

- Numeric – datatype storing numbers (default datatype for numbers)
- Integer – datatype storing integers (without the decimal part)
- Logical – datatype for logical values TRUE/FALSE or 1/0
- Character – datatype for text (single sings and longer text sequences)
- Date – datatype for storing dates (looks like text, stored as a number)

HELPFUL FUNCTIONS

Checking the type of the data:

```
class()
```

Checking if given variable belongs to a specific datatype:

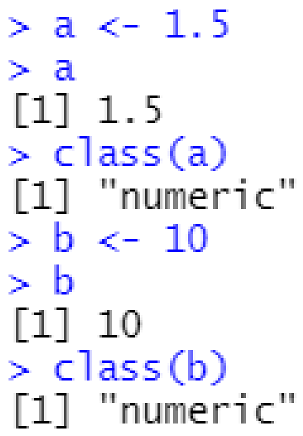
```
is.numeric() is.integer()
  is.logical() is.character()
```

Converting variable to different datatype:

```
as.numeric() as.integer()
  as.logical() as.character()
  as.Date()
```

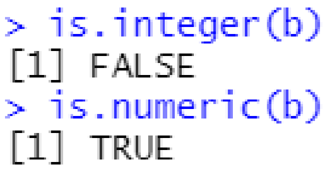
NUMERIC

Basic datatype for storing numerical data



```
> a <- 1.5
> a
[1] 1.5
> class(a)
[1] "numeric"
> b <- 10
> b
[1] 10
> class(b)
[1] "numeric"
```

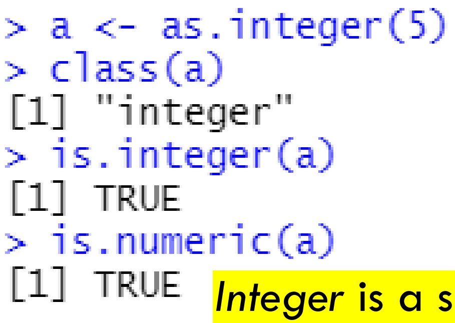
Important! Even if the saved value will include only an integer part (e.g. `b <- 10`), R will store it in a default type *numeric*.



```
> is.integer(b)
[1] FALSE
> is.numeric(b)
[1] TRUE
```

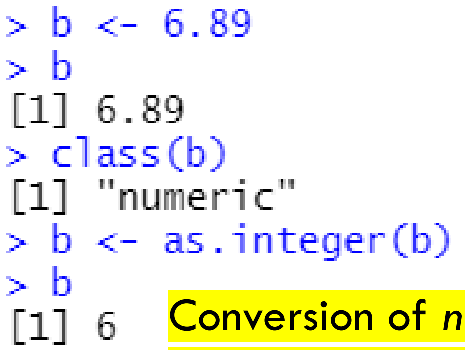
INTEGER

Datatype for numerical data, storing only the integers (without the decimal part). This is a special version (subclass) of the *numeric* class. In order to create a variable with *integer* class, you need to "force it" onto R while creating the variable or convert already existing variable with *numeric* type.



```
> a <- as.integer(5)
> class(a)
[1] "integer"
> is.integer(a)
[1] TRUE
> is.numeric(a)
[1] TRUE
```

Integer is a subclass of the *numeric* type

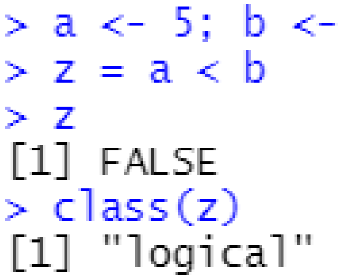


```
> b <- 6.89
> b
[1] 6.89
> class(b)
[1] "numeric"
> b <- as.integer(b)
> b
[1] 6
```

Conversion of *numeric* variable to *integer* deletes the decimal part of the number.

LOGICAL

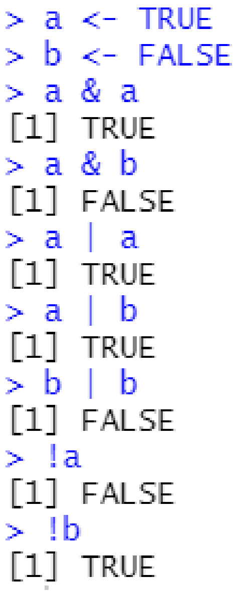
Datatype storing logical information true-false. In R there are alternative ways of writing the logical values: TRUE – FALSE; T – F; 1 – 0.



```
> a <- 5; b <- 3
> z = a < b
> z
[1] FALSE
> class(z)
[1] "logical"
```

Logical operators in R:

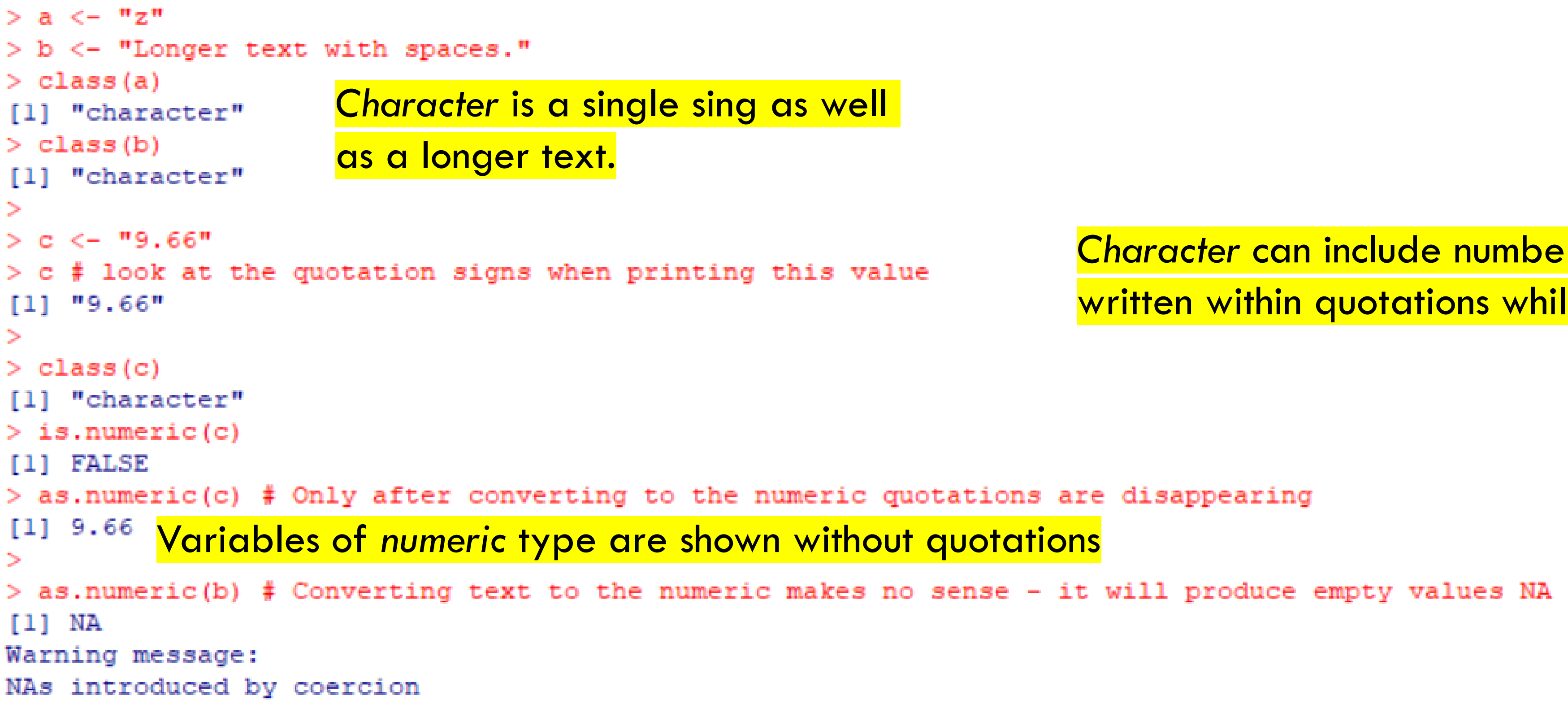
- & (and) conjunction
- | (or) alternative
- ! (not) negation
- == comparison



```
> a <- TRUE
> b <- FALSE
> a & a
[1] TRUE
> a & b
[1] FALSE
> a | a
[1] TRUE
> a | b
[1] TRUE
> b | b
[1] FALSE
> !a
[1] FALSE
> !b
[1] TRUE
```

CHARACTER

Datatype for storing text values. Character data in R are shown within quotation symbols.



```
> a <- "z"
> b <- "Longer text with spaces."
> class(a)
[1] "character"
> class(b)
[1] "character"
>
> c <- "9.66"
> c # look at the quotation signs when printing this value
[1] "9.66"
>
> class(c)
[1] "character"
> is.numeric(c)
[1] FALSE
> as.numeric(c) # Only after converting to the numeric quotations are disappearing
[1] 9.66
>
> as.numeric(b) # Converting text to the numeric makes no sense - it will produce empty values NA
[1] NA
Warning message:
NAs introduced by coercion
```

Character is a single sing as well as a longer text.

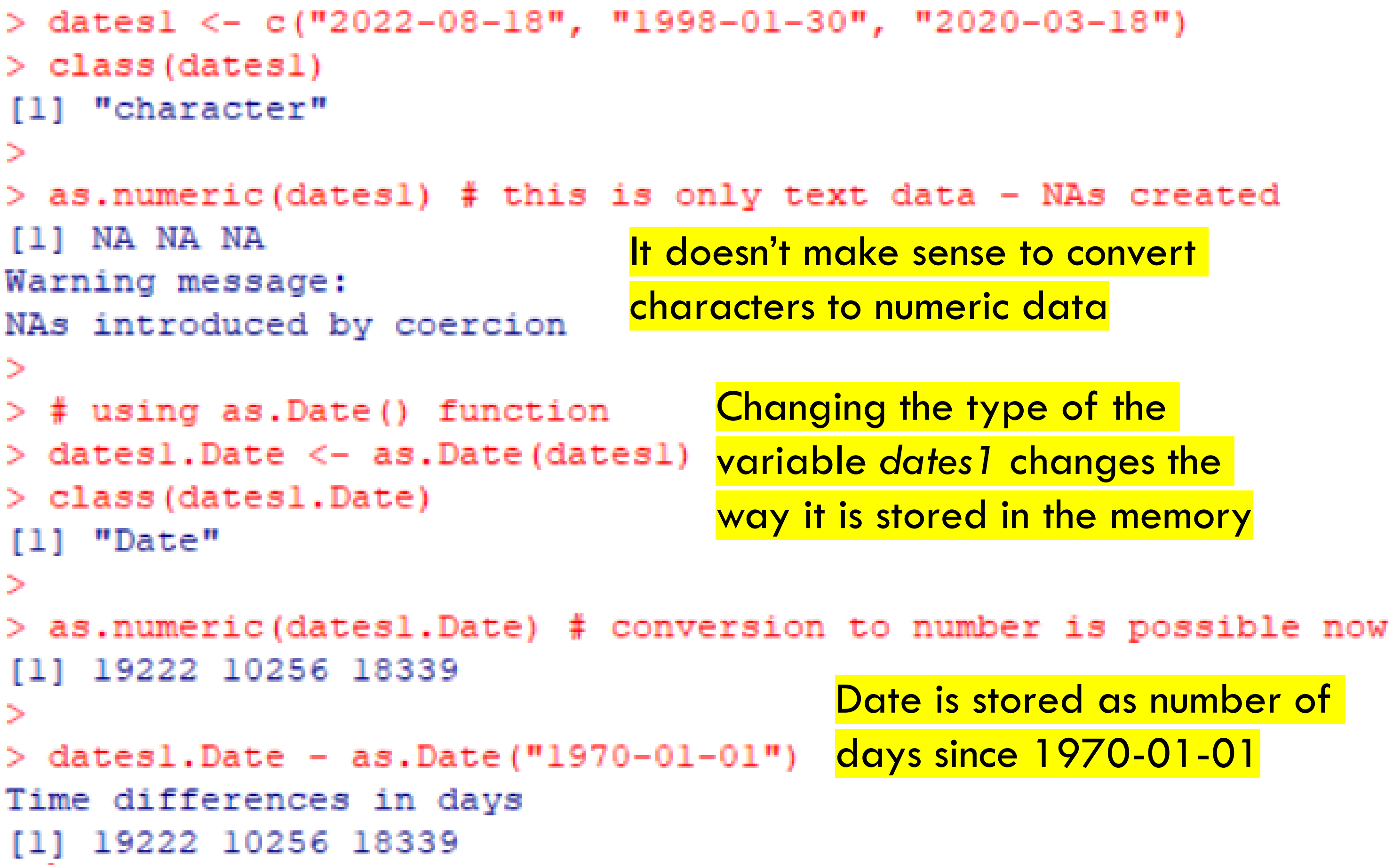
Character can include numbers (if the values are written within quotations while creating a variable).

Variables of *numeric* type are shown without quotations

DATE

We have many approaches in R to process dates.

The standard one comes from basic R and uses function as.Date() to convert strings to dates. Dates are stored as the number of days since 1970-01-01, with negative values for earlier dates. This format stores date-only data.



```
> dates1 <- c("2022-08-18", "1998-01-30", "2020-03-18")
> class(dates1)
[1] "character"
>
> as.numeric(dates1) # this is only text data - NAs created
[1] NA NA NA
Warning message:
NAs introduced by coercion
>
> # using as.Date() function
> dates1.Date <- as.Date(dates1)
> class(dates1.Date)
[1] "Date"
>
> as.numeric(dates1.Date) # conversion to number is possible now
[1] 19222 10256 18339
>
> dates1.Date - as.Date("1970-01-01")
Time differences in days
[1] 19222 10256 18339
```

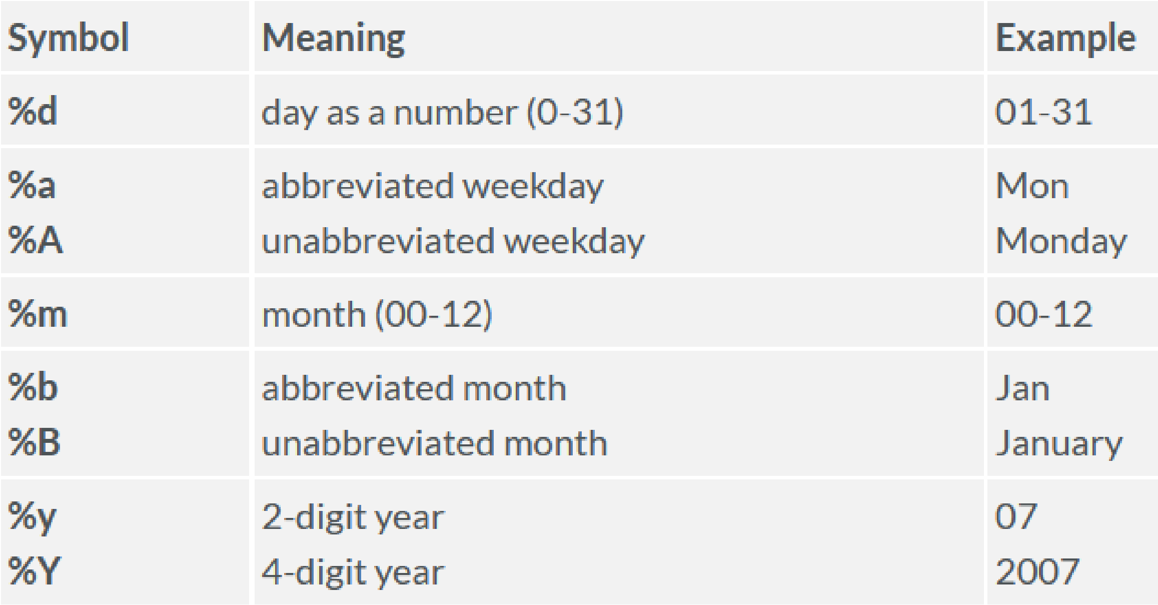
It doesn’t make sense to convert characters to numeric data

Changing the type of the variable *dates1* changes the way it is stored in the memory

Date is stored as number of days since 1970-01-01

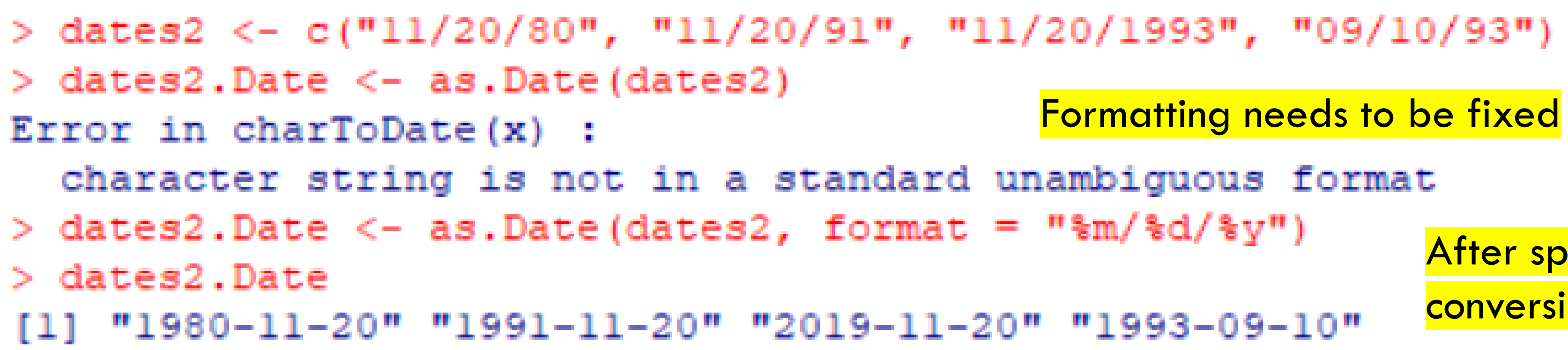
AS.DATE() FORMATTING

Dates need to be formatted in a way that is recognizable by as.Date() function. If dates are formatted differently, we need to specify the format in the additional argument.



Symbol	Meaning	Example
%d	day as a number (0-31)	01-31
%a %A	abbreviated weekday unabbreviated weekday	Mon Monday
%m	month (00-12)	00-12
%b %B	abbreviated month unabbreviated month	Jan January
%y %Y	2-digit year 4-digit year	07 2007

Source: https://www.statmethods.net/input/dates.html



```
> dates2 <- c("11/20/80", "11/20/91", "11/20/1993", "09/10/93")
> dates2.Date <- as.Date(dates2)
Error in charToDate(x) :
  character string is not in a standard unambiguous format
> dates2.Date <- as.Date(dates2, format = "%m/%d/%y")
> dates2.Date
[1] "1980-11-20" "1991-11-20" "2019-11-20" "1993-09-10"
```

Formatting needs to be fixed

After specifying *format* the conversion was successful

ADVANCED DATE PROCESSING

For more advanced date processing in R check classes POSIX (these allow for processing date-time data with time zone corrections and are commonly used for time series analysis of stock data).

Usage of as.POSIX*:
https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/as.POSIX*

Formats:
https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/strptime

DATA STRUCTURES IN R

DATA STRUCTURES

Data structures are a way to store data inside a computer's memory. Depending on the data type and their future usage, data can be stored in different structures.

Structures can be implemented within the programming language in different ways, which may influence the speed of calculations or memory usage.

In R we have a few basic data types:

- Vector
- Matrix
- List
- Data frame

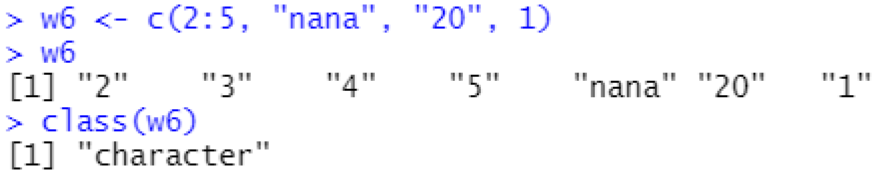
VECTOR

Vector is a sequence of data of the same type (class). Vector is the basic data structure in R. Every single variable in R is just a vector of length one.

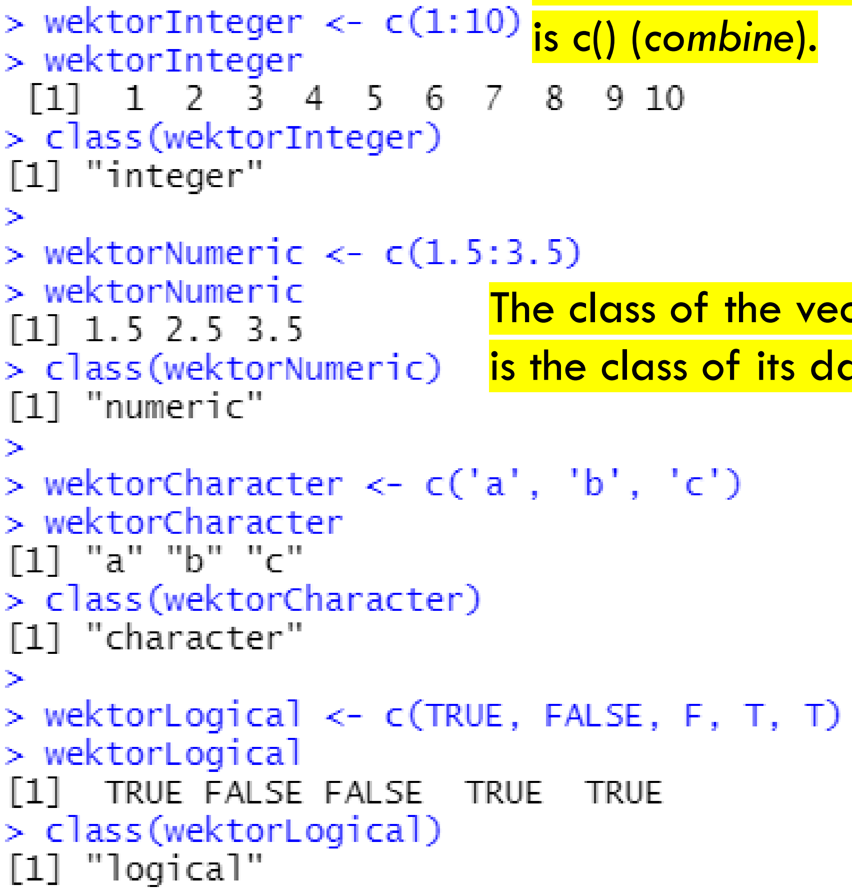
CREATING VECTORS

Basic command for creating vectors is c() c() (*combine*). Helpful functions are also rep() (*replicate*), seq() (*sequence*).

Because we have added elements of type character, all the elements inside the vector have the same class.



```
> w6 <- c(2:5, "nana", "20", 1)
> w6
[1] "2"    "3"    "4"    "5"    "nana" "20"   "1"
> class(w6)
[1] "character"
```



```
> wektorInteger <- c(1:10)
> wektorInteger
 [1]  1  2  3  4  5  6  7  8  9 10
> class(wektorInteger)
[1] "integer"
>
> wektorNumeric <- c(1.5:3.5)
> wektorNumeric
[1] 1.5 2.5 3.5
> class(wektorNumeric)
[1] "numeric"
>
> wektorCharacter <- c('a', 'b', 'c')
> wektorCharacter
[1] "a" "b" "c"
> class(wektorCharacter)
[1] "character"
>
> wektorLogical <- c(TRUE, FALSE, F, T, T)
> wektorLogical
[1]  TRUE FALSE FALSE  TRUE  TRUE
> class(wektorLogical)
[1] "logical"
```

Basic command creating a vector is c() (*combine*).

The class of the vector is the class of its data.

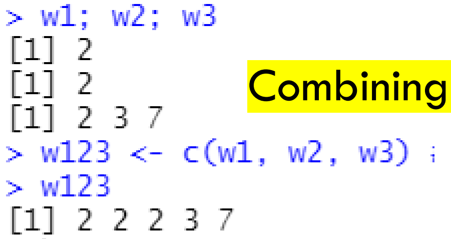
OPERATIONS ON VECTORS

Function c() allows for creating vectors and for merging them.

We can do arithmetical operations on vectors (multiplying and dividing, adding and subtracting etc.).

We can add vectors together:

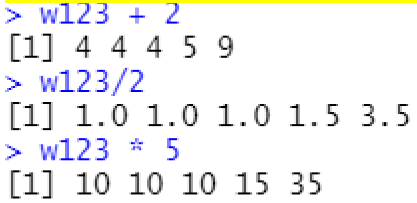
- when they have the same length (x1,x2) + (y1,y2) = (x1+y1, x2+y2)
- when the length of the longer vector is a multiplication of the shorter one (x1,x2)+(y1,y2,y3,y4) = (y1+x1, y2+x2, y3+x1, y4+x2) – recycling rule for vectors



```
> w1; w2; w3
[1] 2
[1] 2
[1] 2 3 7
> w123 <- c(w1, w2, w3)
> w123
[1] 2 2 2 3 7
```

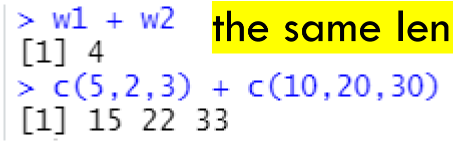
Combining vectors

Arithmetical operations

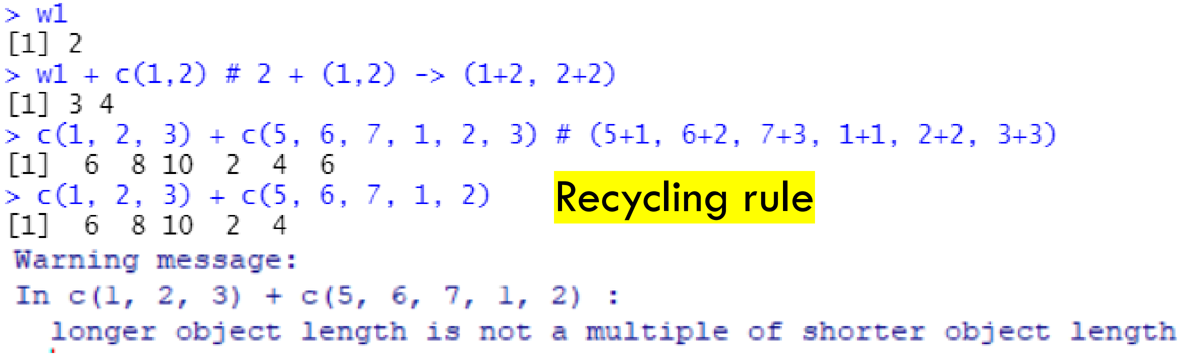


```
> w123 + 2
[1] 4 4 4 5 9
> w123/2
[1] 1.0 1.0 1.0 1.5 3.5
> w123 * 5
[1] 10 10 10 15 35
```

Adding vectors of the same length



```
> w1 + w2
[1] 4
> c(5,2,3) + c(10,20,30)
[1] 15 22 33
```



```
> w1
[1] 2
> w1 + c(1,2) # 2 + (1,2) -> (1+2, 2+2)
[1] 3 4
> c(1, 2, 3) + c(5, 6, 7, 1, 2, 3) # (5+1, 6+2, 7+3, 1+1, 2+2, 3+3)
[1]  6  8 10  2  4  6
> c(1, 2, 3) + c(5, 6, 7, 1, 2)
[1]  6  8 10  2  4
Warning message:
In c(1, 2, 3) + c(5, 6, 7, 1, 2) :
  longer object length is not a multiple of shorter object length
```

Recycling rule

INDEXING OF VECTORS

Using indexes we can take given elements from the vectors.

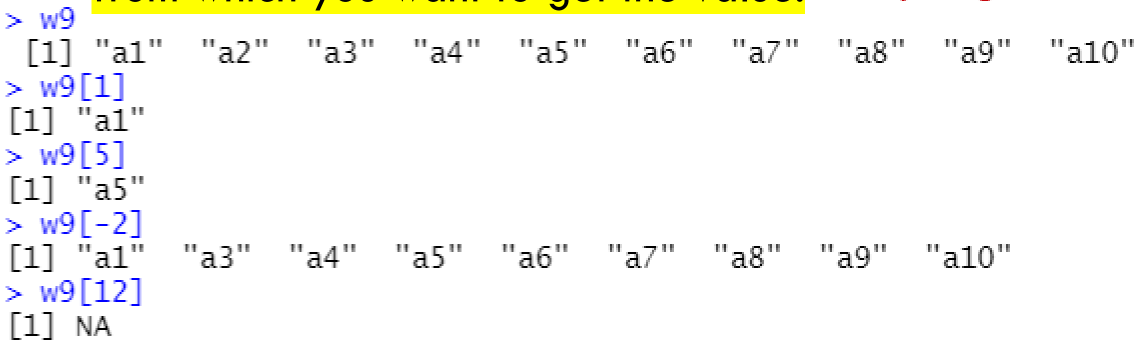
Indexing in R starts from 1!

Indexing in vectors works with squared brackets `vectorName[positionNumber]`.

It is possible to index with negative numbers – it will make R omit a specific element, for which the index was negated.

Trying to get a value outside of the index range will return an empty value NA.

Examples of indexes – show the position from which you want to get the value.



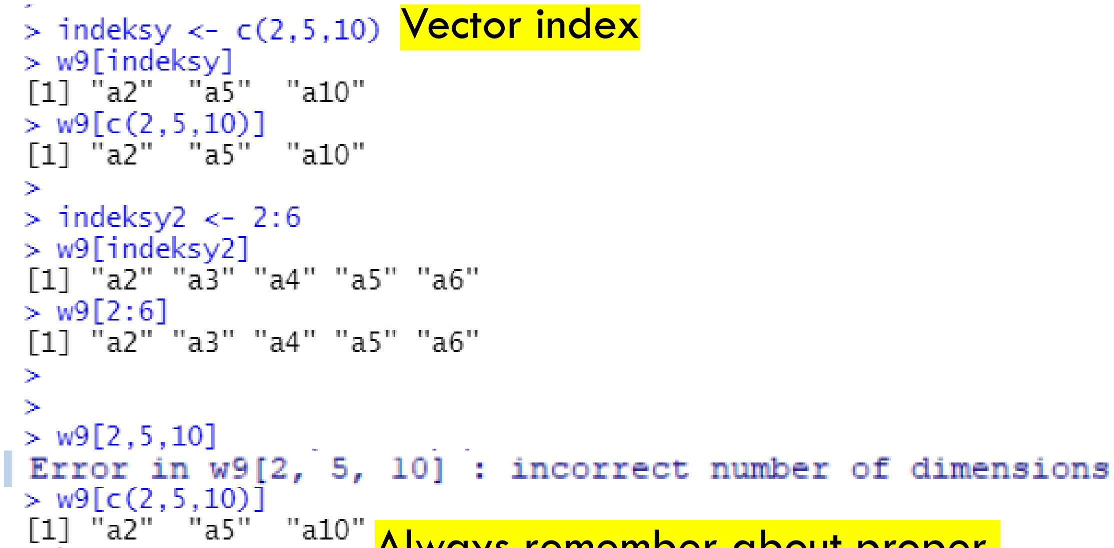
```
> w9
 [1] "a1"  "a2"  "a3"  "a4"  "a5"  "a6"  "a7"  "a8"  "a9"  "a10"
> w9[1]
[1] "a1"
> w9[5]
[1] "a5"
> w9[-2]
[1] "a1"  "a3"  "a4"  "a5"  "a6"  "a7"  "a8"  "a9"  "a10"
> w9[12]
[1] NA
```

INDEXING IN R STARTS FROM 1!!!

INDEXING WITH VECTORS

We can also take many values at once using a vector of indexes.

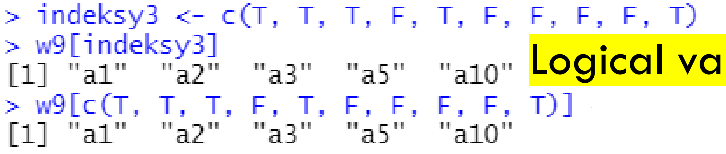
Additionally, one can use indexing by logical values – one needs to declare a vector with TRUE/FALSE values with the same length as the analyzed vector. Using that logical vector we specify which elements we would like to show (TRUE) and which should be omitted (FALSE).



```
> indeksy <- c(2,5,10)
> w9[indeksy]
[1] "a2"  "a5"  "a10"
> w9[c(2,5,10)]
[1] "a2"  "a5"  "a10"
>
> indeksy2 <- 2:6
> w9[indeksy2]
[1] "a2" "a3" "a4" "a5" "a6"
> w9[2:6]
[1] "a2" "a3" "a4" "a5" "a6"
>
>
> w9[2,5,10]
Error in w9[2, 5, 10] : incorrect number of dimensions
> w9[c(2,5,10)]
[1] "a2"  "a5"  "a10"
```

Vector index

Always remember about proper notation (function c() is a safe option)



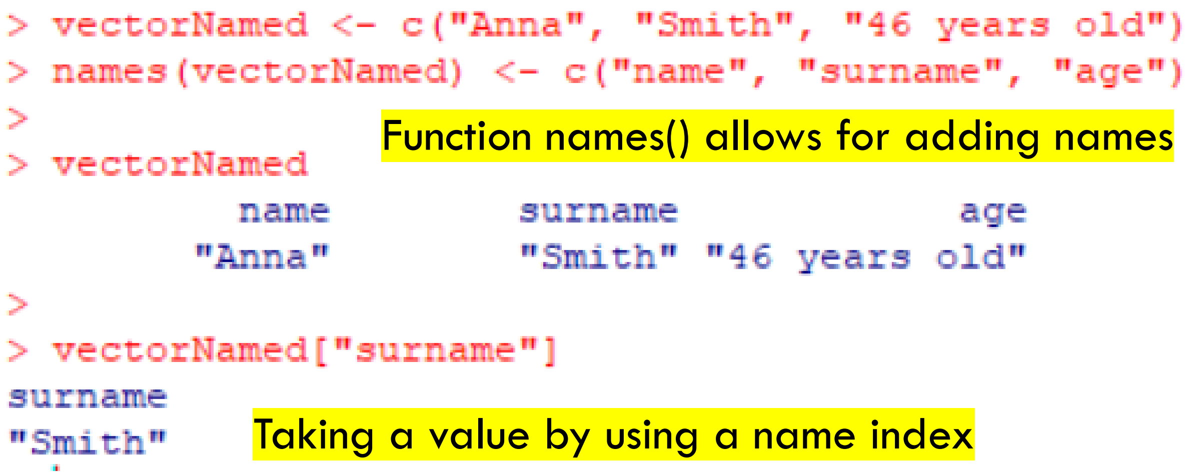
```
> indeksy3 <- c(T, T, T, F, T, F, F, F, F, T)
> w9[indeksy3]
[1] "a1"  "a2"  "a3"  "a5"  "a10"
> w9[c(T, T, T, F, T, F, F, F, F, T)]
[1] "a1"  "a2"  "a3"  "a5"  "a10"
```

Logical values as indexes

INDEXING WITH NAMES

Values in vectors can be names, and then used as indexes.

Vector of names must have the same length as the named vector.

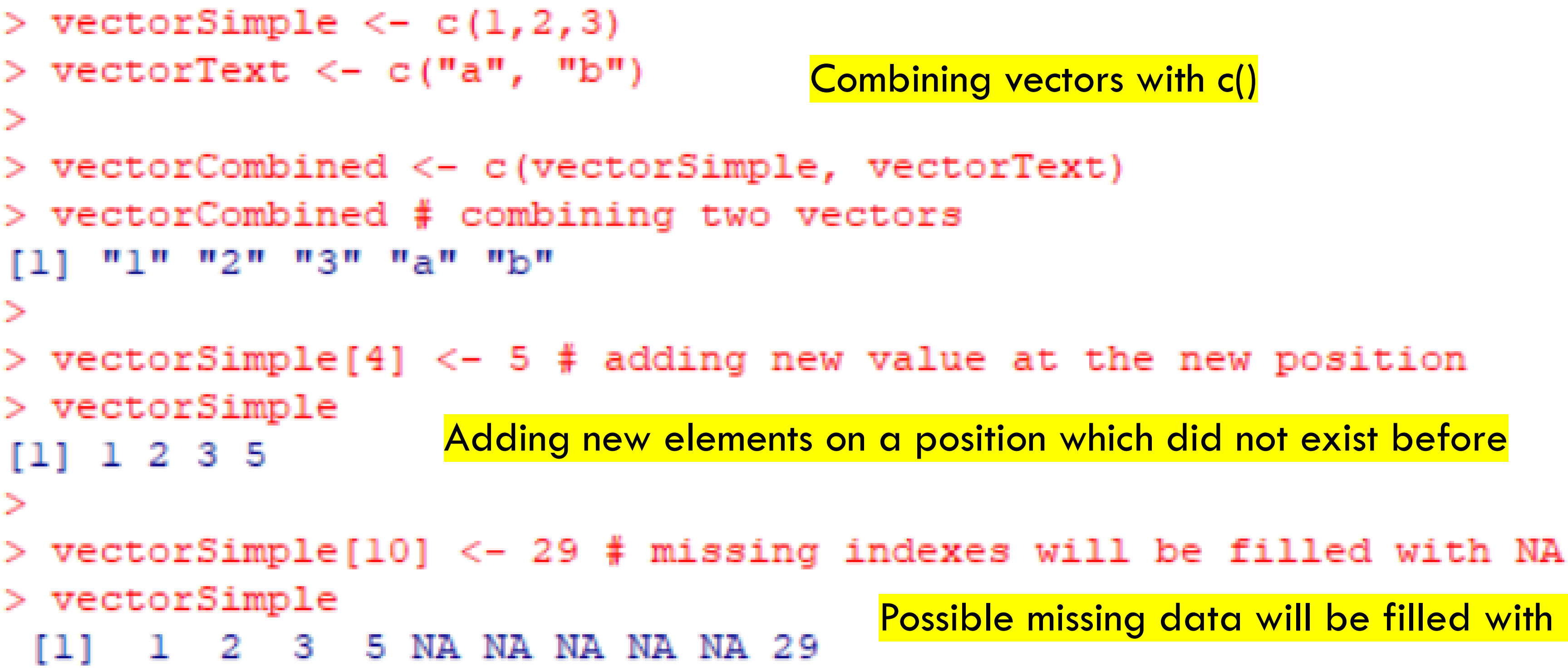


```
> vectorNamed <- c("Anna", "Smith", "46 years old")
> names(vectorNamed) <- c("name", "surname", "age")
>
> vectorNamed
          name        surname             age
        "Anna"        "Smith" "46 years old"
>
> vectorNamed["surname"]
surname
"Smith"
```

Function names() allows for adding names

Taking a value by using a name index

MODIFICATION OF VECTORS



```
> vectorSimple <- c(1,2,3)
> vectorText <- c("a", "b")
>
> vectorCombined <- c(vectorSimple, vectorText)
> vectorCombined # combining two vectors
[1] "1" "2" "3" "a" "b"
>
> vectorSimple[4] <- 5 # adding new value at the new position
> vectorSimple
[1] 1 2 3 5
>
> vectorSimple[10] <- 29 # missing indexes will be filled with NA
> vectorSimple
 [1]  1  2  3  5 NA NA NA NA NA 29
```

Combining vectors with c()

Adding new elements on a position which did not exist before

Possible missing data will be filled with NA