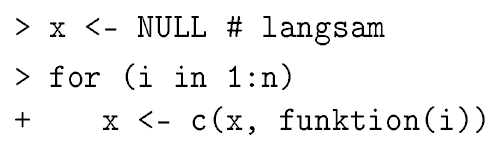
**Iterations (or Loops)**

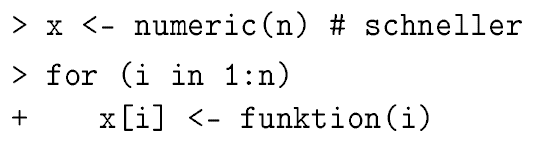
* are another tool to reducing duplications
* iterations help you when you need **to do the same thing on different inputs**, like different columns or different datasets
* **a Loop is code that tells a program like R to repeat a certain chunk of code several times,   
  with different values of an iterator variable that changes for every run of the loop**

**Components of a Loop**

* every Loop has the following 3 components:

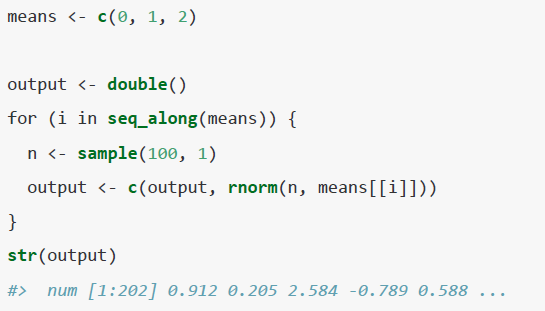
1. **The output:** 
   * every run of the loop provides an output value
   * in very rare cases it will be sufficient to only print these output values of the Loop
   * in most cases we want to save them in an object
   * working with a Loop we have to allocate sufficient space for the output by creating a respective object, **before you start the loop**
   * this is very important for the loop’s efficiency: **if you grow the output iteratively** (e.g. using c()) the loop will be very slow 🡪 in each iteration R then has to copy the data from the previous iterations
   * a general way of **creating an empty vector of given length is** > vector()
     + this function has 2 arguments:
       - datatype: “logical”, “integer”, “double”, “character” etc.
       - length (can be left empty)
     + mit vector(mode = “list”, length = <n>) lässt sich außerdem eine leere Liste erstellen
   * zur Erzeugung leerer Vektoren bestimmter Datentypen dienen außerdem:   
     logical(), character(), numeric(), complex() etc.





* + im letzten Beispiel nutzen wir i als Index für x und weisen jedem Index i einen Wert aus Funktion(i) zu
* **Unknown output length**
  + there might occur situations in which you do not know the length of the output vector
  + then it is a good idea to create a list first and then unlist to one vector afterwards
  + ***do not avoid stating the length by letting the vector grow progressively:***
* in the following example we want to create random numbers of normal distribution, taking three different means

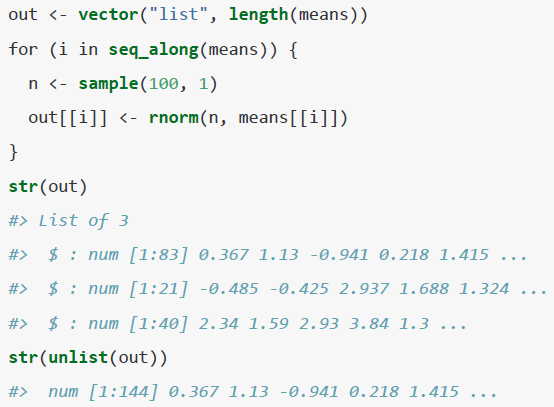
**Don’t:**



**Do:**

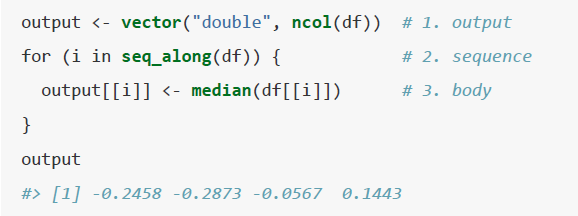


# of list elements



* instead of calculating the # of numbers in the end results, we simply create a list
* **this pattern applies to other examples too:**
  + you might want to create **a long string:** 
    - instead of pasting each iteration together, save the output in a character vector   
      and then combine them into a single string, e.g. using > paste(output, collapse = “”)
  + you might be **generating a big df:** 
    - instead of sequentially rbind() ing in each iteration, **save the output in a list   
      and the use e.g.** >dplyr::bind\_rows(output) **to combine the output into a single data fame**

1. **The sequence:** 
   * determines the varying input over which to perform the iterations
2. **The body:** 
   * the code that is actually computed repeatedly



**base R’s For Loops**

* **Hierbei wird die Anzahl der Iterationen vor Beginn der Schleife genau festgelegt**
* **For Loops iterate over a vector, list or data frame**

1. **General syntax:**

**for**(<iterator variable i> **in** <vector M>){

**expr**

}

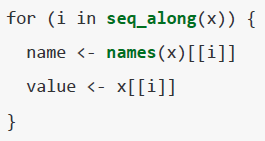
**“ Wiederhole Ausdruck für jedes i in M“**

* iterator variable: determines the varying input over which to perform the iterations
  + this object will change for each iteration of the loop
* vector M: specifies all values that the loop object will take over the loop
  + > seq\_along(<input vector>) is a useful function in this context
    - it creates a sequence of indices given the length of a vector
    - 
  + if one wants to run a loop over very specific values, one can use c() 2
* **expr**: expression that will be evaluated/computed iteratively
  + **we must refer to the iterator variable in the expression**

**Details on the iterator variable:**

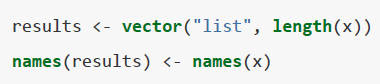
**There are overall 3 ways to iterate over a vector:**

**1) Loop over the numeric indices:** > for(i in seq\_along(<M>)

* **seq\_along** creates an integer vector of indices for respective objective
* values are then extracted with x[[i]]
* iterating over indices is the **most** **general form, because given the position you can extract both over name and value:**

**2) Loop over names:** > for(nm in names(<M>))

* names is the function I already now, providing the names of a named object
* values are then extracted with x[[nm]]
* **this is useful if you want to use the name in a plot title or file name**
* if you are creating named output, make sure that the results from the loop have the same name as the initial object:



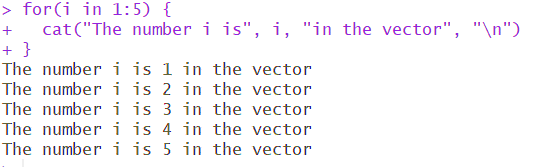
**3) Loop over the elements in M:** > for(x in <M>)

* this is most useful if you only care about the side-effects like plotting or saving a file, because otherwise it is difficult to save the output efficiently

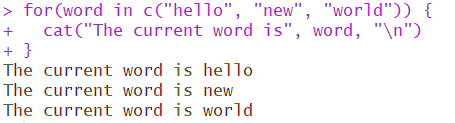
1. **Introductory examples**

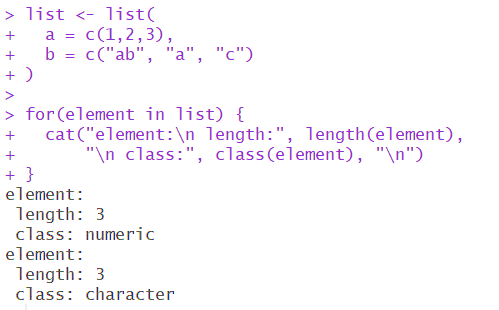
* note that in all examples we insert \n to have multiple lines; otherwise the result will be only one string

**Simple example with numeric vector:**

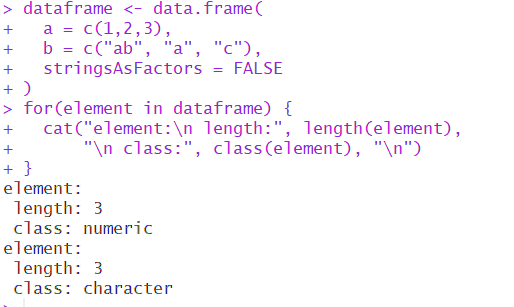


**Simple example with non-numeric vector**



**Simple example with list:**

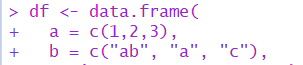
**Simple example w/ data frame:**

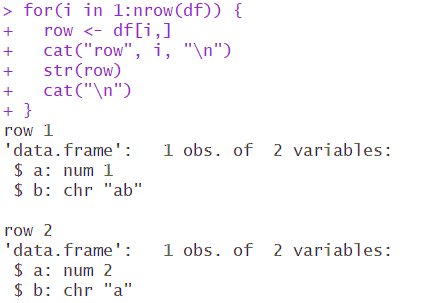


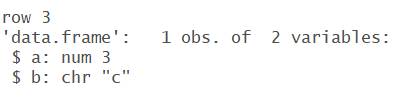
**Remarks on using a data frame for a loop:**

* by default, R will apply the iteration column-wise
* however, in many case we want to iterate over a data frame row by **using the row index as an iterator variable**

Example w/ data frame row-wise iteration





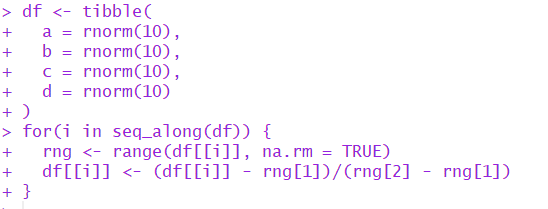


* in this example we determine the structure (with str()) of each row of the data frame

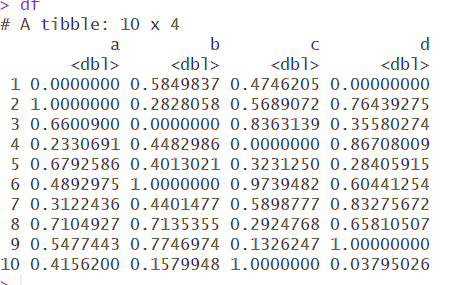
1. **How a For Loop works**

* **so how does a For Loop work?**
  + the loop is done **for an iterator variable**
  + this iterator variable is given data by supplying a vector or list in the same line
  + **the exact computation is expressed in curvy brackets**

1. **Modifying an existing object:**

* the only difference is that we refer to an existing object in our Loop Body:
* der folgende Loop normalisiert alle Werte des data frames:   
    
  

**Result:**

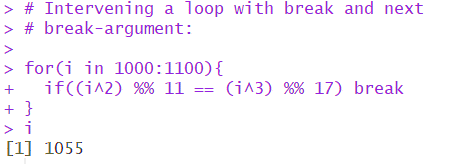


1. **Managing the flow of a For Loop**

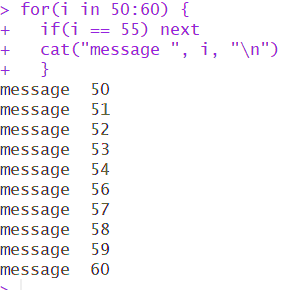
* sometimes it is useful to intervene in a For Loop
* in each iteration we can:
  + interrupt the for loop (> break argument)
    - this is useful if e.g. you only need to have one element that satisfies a condition (see example)
  + skip the current iteration (> next argument)
  + or do nothing and finish the loop (as usual)

**Example for interrupting a For Loop using > break**

* in this example we are trying to find numbers (i) between 1000 and 1100 for which the following condition becomes true:   
  **i^2 %% 11 == i^3 %% 17**
* the loop should stop if we find one number for which this holds true
* **note:** 
  + R does **not automatically print the results**
  + **once a solution is found and the loop stops, the last value of i is automatically preserved in the current environment and can be called**

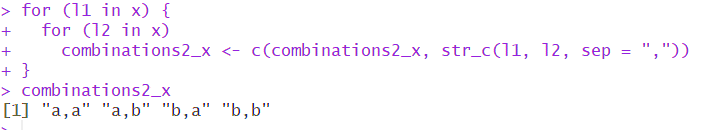
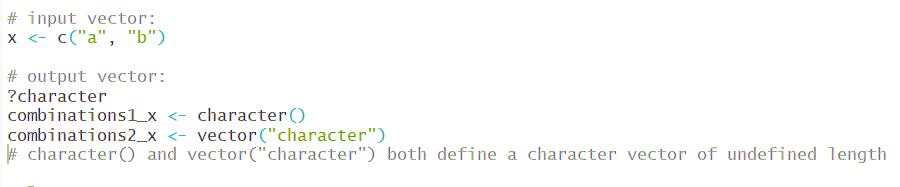
**Example for skipping an iteration:**



1. **Creating Nested For Loops**

* the expression in a For Loop can by anything, incl. **another For Loop**
* might be useful if we, for example, we want to exhaust all permutations of the elements in a vector

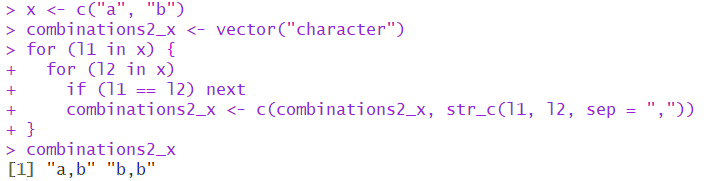
**Example:**

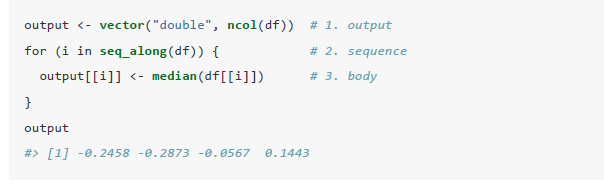


**note: for a nested loop it is not enough to only use concatenating argument (in this case str\_c);**

* **one has to use “double-concatenating” with c()**

**Example: nested loop and next-argument**



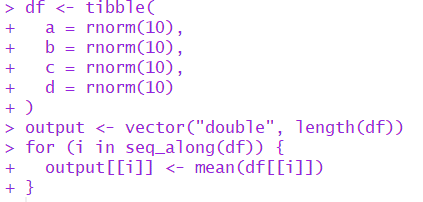


**For Loops vs functionals:**

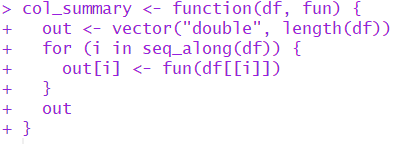
* R is a functional language; this means, **it is possible to wrap up For Loops in a function, and call that function instead of using the for loop directly**
* this is especially useful, if you want to use a loop frequently, on changing objects

**Example:**

* assume you want to calculate the mean, median and standard deviation for each column in a data frame
* you could create following loop for each statistical measurement:



* but then one would have to copy and adjust this loop for every different data frame provided, as the input data is not seen as a variable, but fixed in a loop
* **so the better solution is to use following function, with the df and the stats being the functional arguments:**



**base R’s While Loop [rarely used]**

* While Loops are rarely used, which is why I cover them only very briefly
* this loop continues as long / **while a certain conditions still holds**
* it terminates [ende] as soon as this condition does not hold anymore

**General syntax:**

while(<Bedingung>) {

Ausdruck

**}  
“Wiederhole Ausdruck, solange Bedingung erfüllt ist”**

**Simple example:**



* “while x is <= 4, print out the numbers (represented by cat), separated by ws”
* if we would have not included x <- x+1, the loop would go forever: we have assigned value 0 to x, so it would always be <= 4