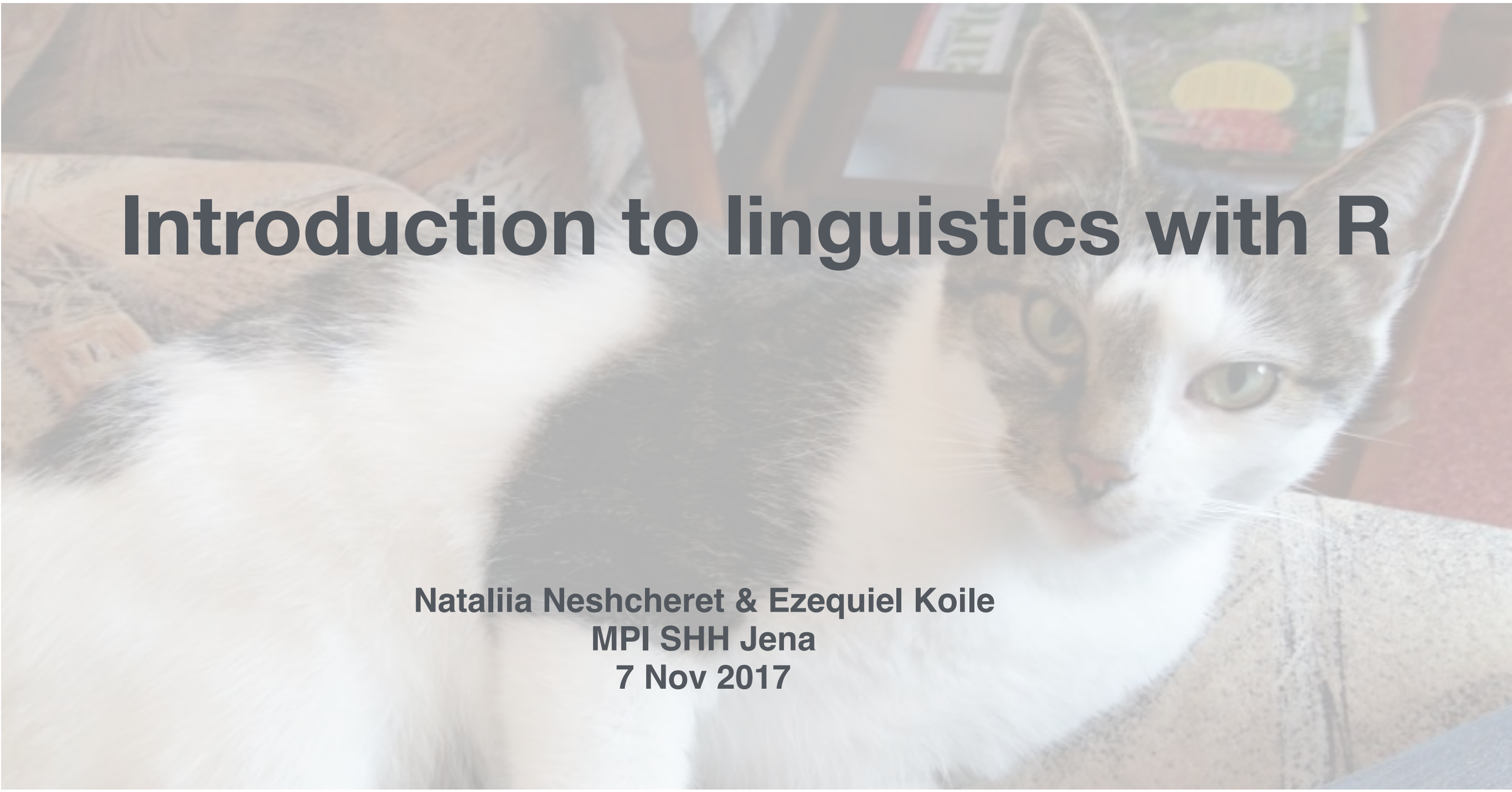


Introduction to linguistics with R

Nataliia Neshcheret & Ezequiel Koile
MPI SHH Jena
7 Nov 2017

What is R and why R?

Advantages

- open source: **free** of charge, constantly under development (you can write your own package too!)

- offers packages for **statistics** and **visualisation**

Drawbacks

- hard at the beginning, getting easier later on

- no buttons to push, you have to **communicate with R** in its **language**

Why are you in this course?

- Because things you like include:
  - languages
  - trees
  - the challenge to combine the upper two

Why are you in this course?

- Because things you like include:
  - languages
  - trees
  - the challenge to combine the upper two
- but mainly because tomorrow comes Annemarie with a block of code like:

Why are you in this course?

- Because things you like include:
  - languages
  - trees
  - the challenge to combine the upper two
- but mainly because tomorrow comes Annemarie with a block of code like:

```
> abc <- read.tree(text = "((A,B),C);") #create a tree with three taxa
> plot.phylo(abc, type="clado") #plot the tree
> tiplabels() #plot the tip labels on the tree
> nodelabels() #plot the node on the tree
> abc$edge #show the matrix
> PN<-read.nexus("pamanyungan.txt")
> colorscheme<-rep("black",2*N)
> for(i in 1:(2*N)){
      if(PN$edge[i,2] < N+2){colorscheme[i]<-rainbow(N+1)[PN$edge[i,2]]}
  }
> plot(PN,edge.color=colorscheme,edge.width=2,show.tip.label=FALSE)
```

Why are you in this course?

- Because things you like include:
  - languages
  - trees
  - the challenge to combine the upper two
- but mainly because tomorrow comes Annemarie with a block of code like:

```
> abc <- read.tree(text = "((A,B),C);") #create a tree with three taxa
> plot.phylo(abc, type="clado") #plot the tree
> tiplabels() #plot the tip labels on the tree
> nodelabels() #plot the node on the tree
> abc$edge #show the matrix
> PN<-read.nexus("pamanyungan.txt")
> colorscheme<-rep("black",2*N)
> for(i in 1:(2*N)){
      if(PN$edge[i,2] < N+2){colorscheme[i]<-rainbow(N+1)[PN$edge[i,2]]}
  }
> plot(PN,edge.color=colorscheme,edge.width=2,show.tip.label=FALSE)
```

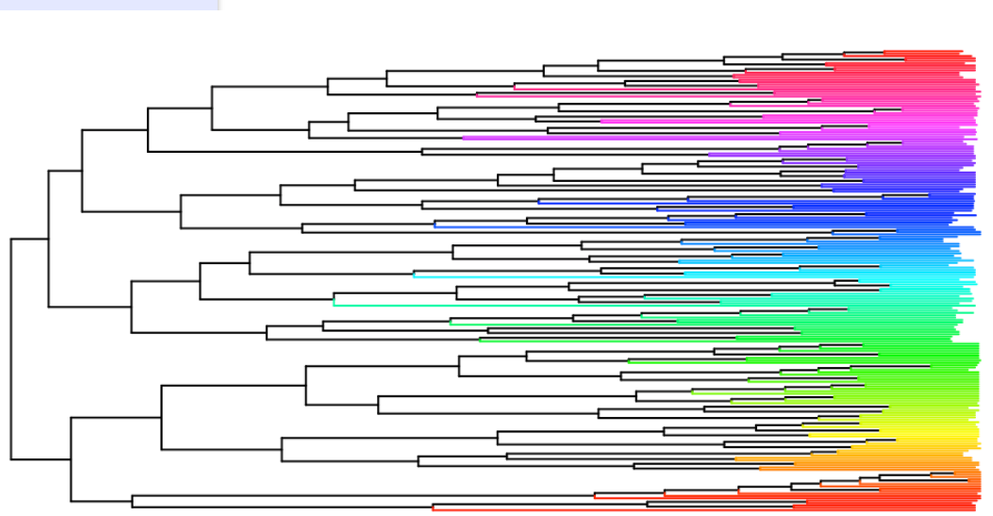
to make a tree like:

Why are you in this course?

- Because things you like include:
  - languages
  - trees
  - the challenge to combine the upper two
- but mainly because tomorrow comes Annemarie with a block of code like:

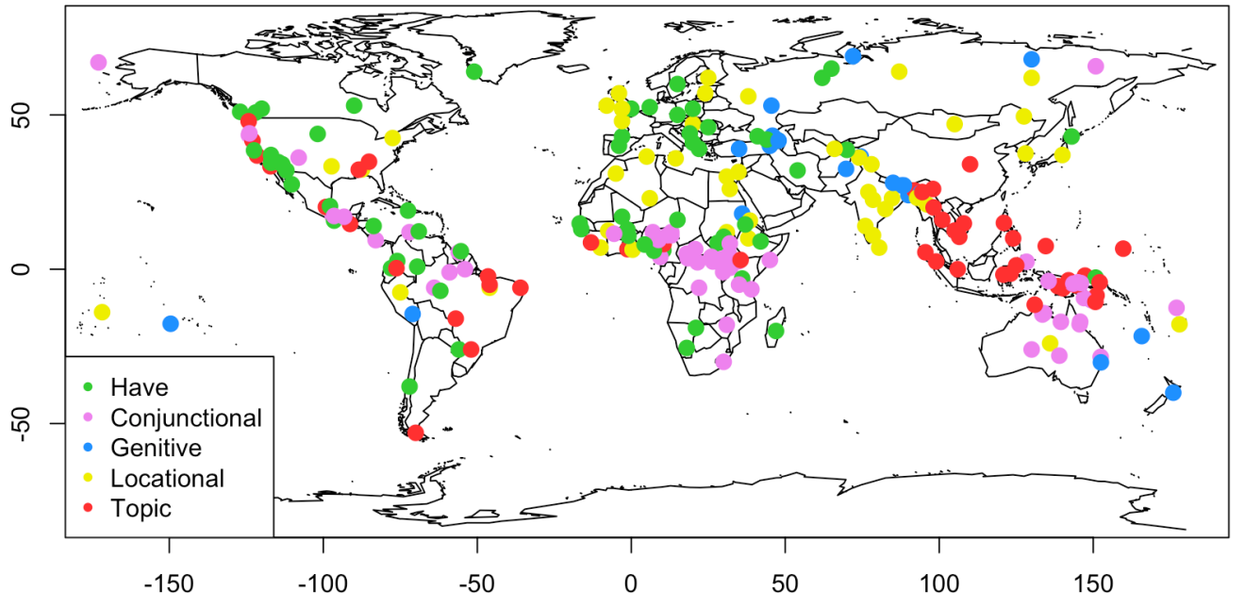
```
> abc <- read.tree(text = "((A,B),C);") #create a tree with three taxa
> plot.phylo(abc, type="clado") #plot the tree
> tiplabels() #plot the tip labels on the tree
> nodelabels() #plot the node on the tree
> abc$edge #show the matrix
> PN<-read.nexus("pamanyungan.txt")
> colorscheme<-rep("black",2*N)
> for(i in 1:(2*N)){
      if(PN$edge[i,2] < N+2){colorscheme[i]<-rainbow(N+1)[PN$ed
  }
> plot(PN,edge.color=colorscheme,edge.width=2,show.tip.label=F
```

to make a tree like:

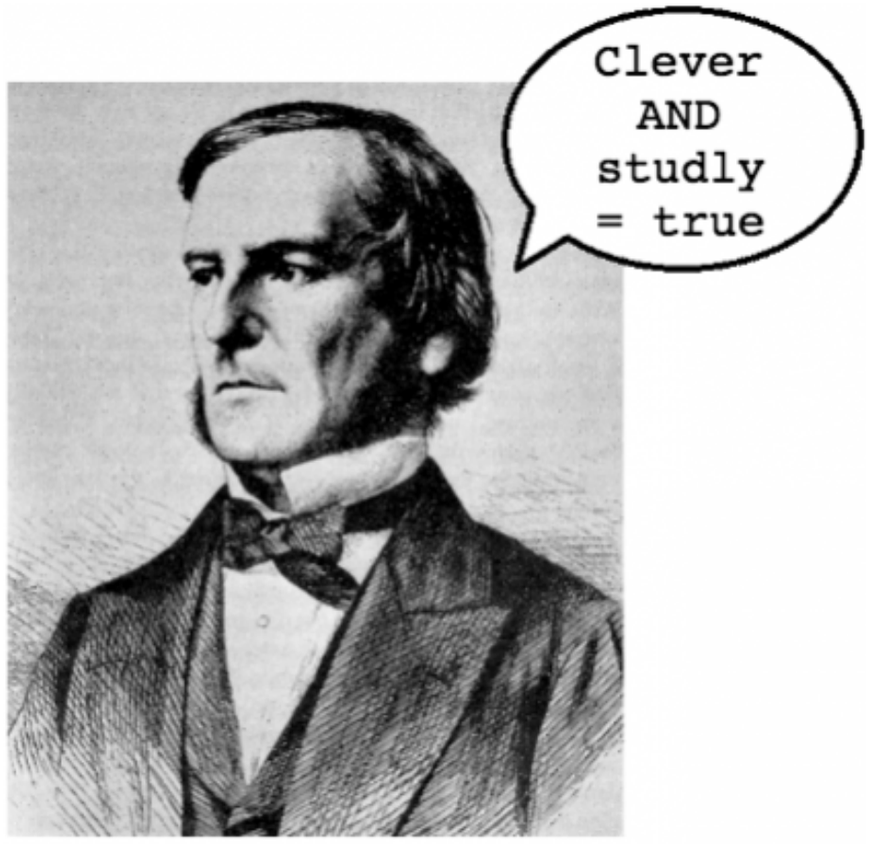


Goals

- R as a calculator
- variable assignment
- functions



- installing packages, loading libraries
- making a map with the packages **`maps`** and **`mapdata`**
- Boolean data type, conditional statements, loops



R layout

RStudio

Go to file/function | Addins | Project: (None)

mini-bayesian-school2017.R

Source on Save | Run | Source

```
1  #some script
2  favourite_languages <- c("Turkic", "Mongolic", "Tungusic",
3                           "Japonic", "Koreanic")
```

Scripts

3:48 (Top Level) R Script

Console ~/Documents/eurasia3angle/conferences/orientalistentag2017/data/

```
S84 +towgs84=0,0,0")
> data = data
> sakhalin_sp
>
> #import dat
> tea<-read.c
> Turkic<-rea
> Mongolic<-r
> Tungusic<-read.csv("Tungusic.csv",sep=";")
```

Console

Environment | History

Import Dataset | List

Global Environment

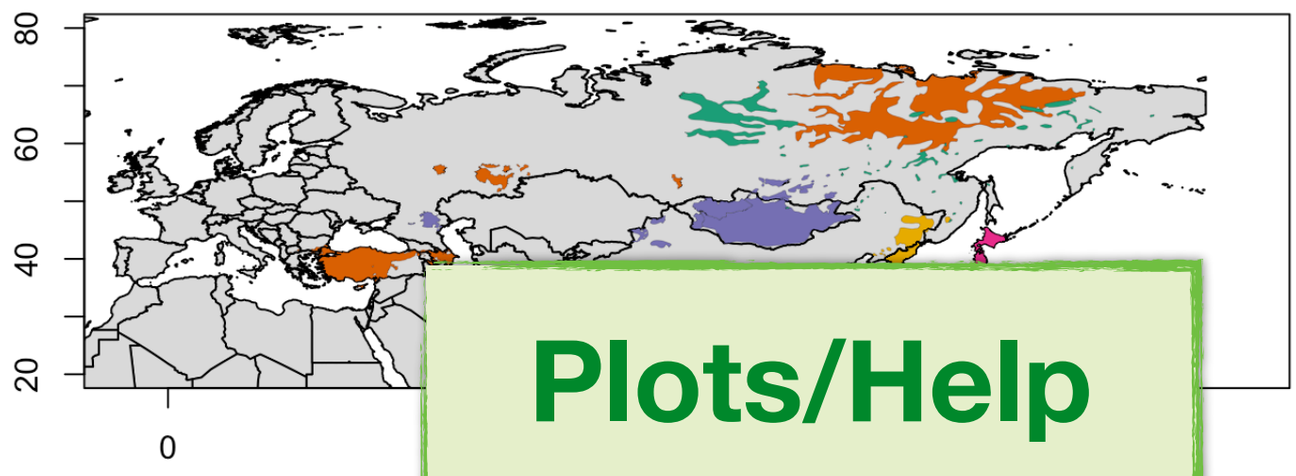
Data

data	
hokkaidoMatr	
hokkaidoTab	
Japonic	
kurilMatr	
kurilTab	
Mongolic	
sakhalinMatr	
sakhalinTab	
tea	17 obs. of 7 variables

Workspace/ History

Files | Plots | Packages | Help | Viewer

Zoom | Export | Publish



Plots/Help

Witzlack 2015

R layout

RStudio

Go to file/function | Addins | Project: (None)

mini-bayesian-school2017.R

Source on Save | Run | Source

```
1  #some script
2  favourite_languages <- c("Turkic", "Mongolic", "Tungusic",
3                           "Japonic", "Koreanic")
```

Scripts

Environment | History

Import Dataset | List

Global Environment

Data

data	1 obs. of 1 variable
hokkaidoMatr	num [1:351, 1:2] 141 141 141 141 141 ...
hokkaidoTab	351 obs. of 3 variables
Japonic	2 obs. of 7 variables
kurilMatr	num [1:87, 1:2] 156 156 156 156 157 ...
kurilTab	87 obs. of 3 variables
Mongolic	5 obs. of 7 variables
sakhalinMatr	num [1:230, 1:2] 144 144 143 143 143 ...
sakhalinTab	230 obs. of 3 variables
tea	17 obs. of 7 variables

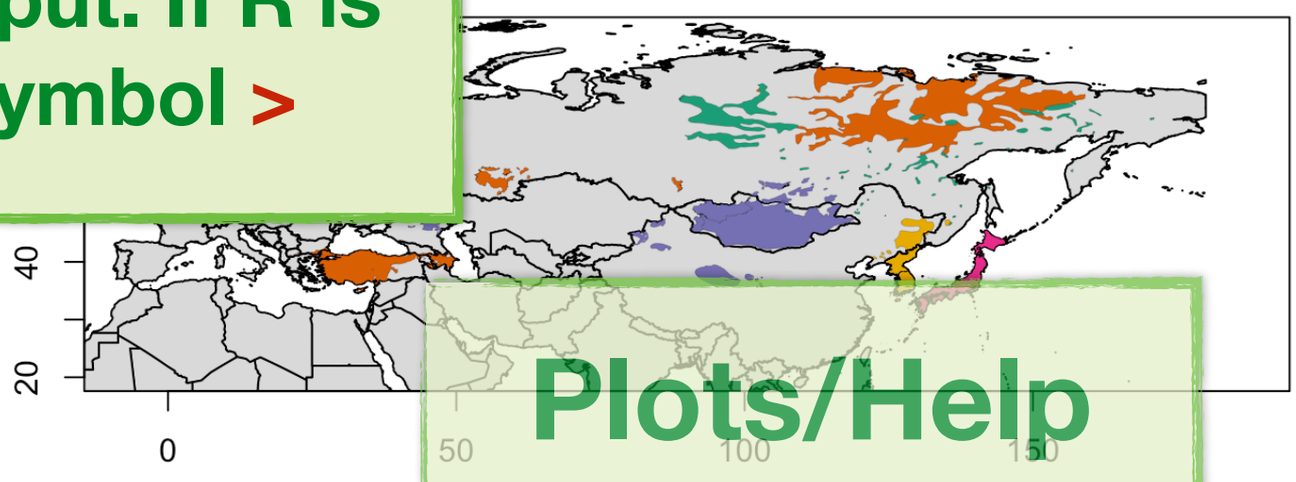
Workspace/ History

You can type in your commands in the console and wait for the output. If R is ready to work, there is a symbol >

```
S84 +towgs84=0,0,0")
> data = data
> sakhalin_sp
>
> #import dat
> tea<-read.c
> Turkic<-rea
> Mongolic<-r
> Tungusic<-read.csv("Tungusic.csv",sep=";")
```

Console

Publish



Plots/Help

Witzlack 2015

Using the console as a calculator

- Type in the console 1+1 and press the Enter button
- Type any other mathematical expression and press Enter
- R does not care about spaces

Addition +

Subtraction -

Multiplication *

Division /

Power ^

```
> 23-1
[1] 22
> 134 - 23*4
[1] 42
> 1+    2
[1] 3
> 2*(6-2)
[1] 8
>
```

Command history

- You can scroll through the command history, execute again or modify an older command

- use the keys ⬆ and ⬇ of the keyboard

- notice how the command reappears at the bottom of the console behind the prompt

- when you reach the command you want to execute again, press Enter

- You cannot modify the text you see above the prompt! Think of it in terms of not being able to change the past

Variable assignment

- if you use particular numbers and other objects often, you can save them in a variable

- with the assignment operator <-

- the names of the variables should not contain any spaces or any number at the beginning

- if a variable contains several words, you can separate them by dots or underscore: `object.example` or `object_example`

```
> d <- 515082
> 45-d
[1] -515037
> d+123
[1] 515205
> d
[1] 515082
```

Functions

- R contains many built-in functions. e.g. `date()`:

```
> date()
> [1] "Fri Nov  3 11:38:05 2017"
```

NameOfTheFunction(argument1, argument2, argument3,…)

- Functions can have 0,1, 2, … obligatory arguments and also optional arguments.

- Think of it as verbs being atransitive, intransitive, transitive, bitransitive and having adjuncts

Witzlack 2015

Functions

- What do the following functions do? (They do not need any arguments)

- `ls()`

- `colors()`

- What does the following one-place function do?

- `sqrt(9)`

- Try out the function `sum()`, it can take many arguments

Functions

- The most popular function: `c()` (concatenate)

Rule of thumb for beginners:

if something does not function, try to build in the concatenate function c()

- `c()` combines arguments to an object that is called **vector**

```
> Pronomina <- c(9773, 1663, 5022, 2500, 691)
> Pronomina
[1] 9773 1663 5022 2500  691
```

Functions

- When is `c()` needed?

- If one wants to execute the same operation with several objects (e.g. multiplication) (=vectorized calculation)

```
> Pronomina <- c(9773, 1663, 5022, 2500, 691)
> Pronomina
[1] 9773 1663 5022 2500  691
> Pronomina/515082*1000
[1] 18.973678  3.228612  9.749904  4.853596  1.341534
```

Functions

- When is `c()` needed?

- if a function takes only one argument, but needs to be applied to several objects.

E.g. the function **`mean()`** takes only one argument, but one normally wants it to calculate the mean of several values

```
> mean(3, 5, 6, 9)
[1] 3
> my_numbers <- c(3, 5, 6, 9)
> mean(my_numbers)
> [1] 5.75
> mean(c(3, 5, 6, 9))
> [1] 5.75
```

this is not what we wanted

this functions!

Task: functions

- The following vowel inventory is given (n=30):

- 21,4,14,5,13,11,15,13,12,12, 21, 12, 11, 5, 17, 18, 20, 22, 16, 17, 25, 7, 20, 15, 4, 10, 10, 10, 17, 18

- Save it as a vector in the variable **vowels**

- use the function `c()` to do so
- check, whether you have all the 30 values with the function `length()`
- calculate the mean with the function `mean()`
- calculate the median with the function `median()`
- find out what the function `table()` does

Task: functions

```
> vowels <- c(21,4,14,5,13,11,15,13,12,12, 21, 12, 11,
5, 17, 18, 20, 22, 16, 17, 25, 7, 20, 15, 4, 10, 10,
10, 17, 18)
> length(vowels)
[1] 30
> mean(vowels)
[1] 13.83333
> median(vowels)
[1] 13.5
> table(vowels)
vowels
 4  5  7 10 11 12 13 14 15 16 17 18 20 21 22 25
 2  2  1  3  2  3  2  1  2  1  3  2  2  2  1  1
```

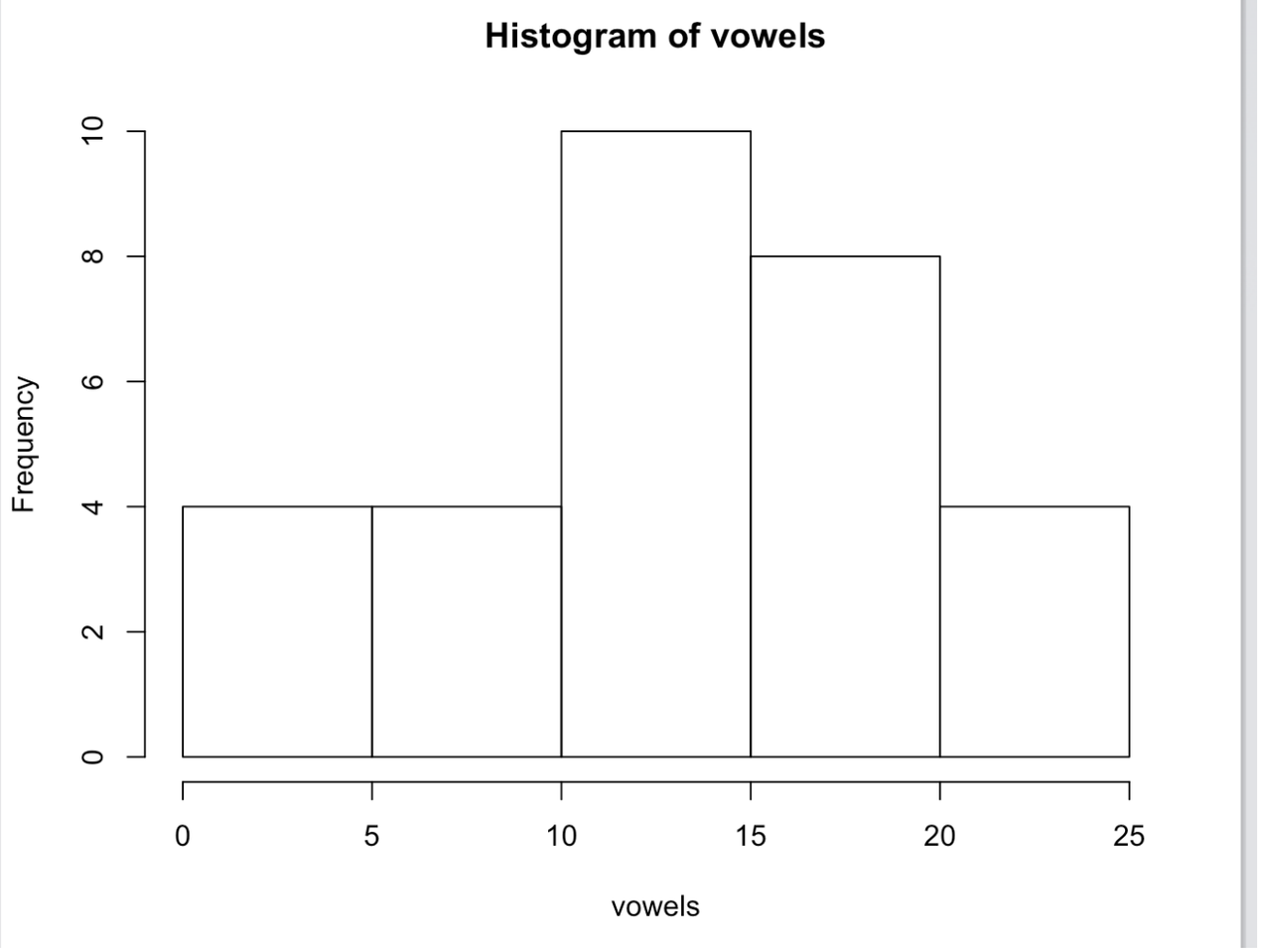
Task: histogram

- Produce a histogram using the function **`hist()`**

RStudio

```
# load the packages
library(maps)
library(mapdata)

# limit the map to central Europe
map(xlim=c(-20,40),ylim = c(40,60))

# limit the map to North-East Asia
map(xlim=c(20,180),ylim = c(30,90))

# limit the map to Germany
map('worldHires','Germany')

# add a point for the MPI
points(11.580738, 50.918549)

# add a colorful point for the MPI
points(11.580738, 50.918549, pch = 20, cex=3, col="tomato")

# add axes to a map
map.axes()
```

```
> vowels <- c(21,4,14,5,13,11,15,13,12,12, 21, 12, 11, 5, 17, 18, 20, 22, 16,
17, 25, 7, 20, 15, 4, 10, 10, 10, 17, 18)
> length(vowels)
[1] 30
> mean(vowels)
[1] 13.83333
> median(vowels)
[1] 13.5
> table(vowels)
vowels
 4  5  7 10 11 12 13 14 15 16 17 18 20 21 22 25
 2  2  1  3  2  3  2  1  2  1  3  2  2  2  1  1
> hist(vowels)
>
```



Task: histogram

- The function `hist()` can take many optional arguments

NameOfTheFunction(argument1, argument2, argument3,…)

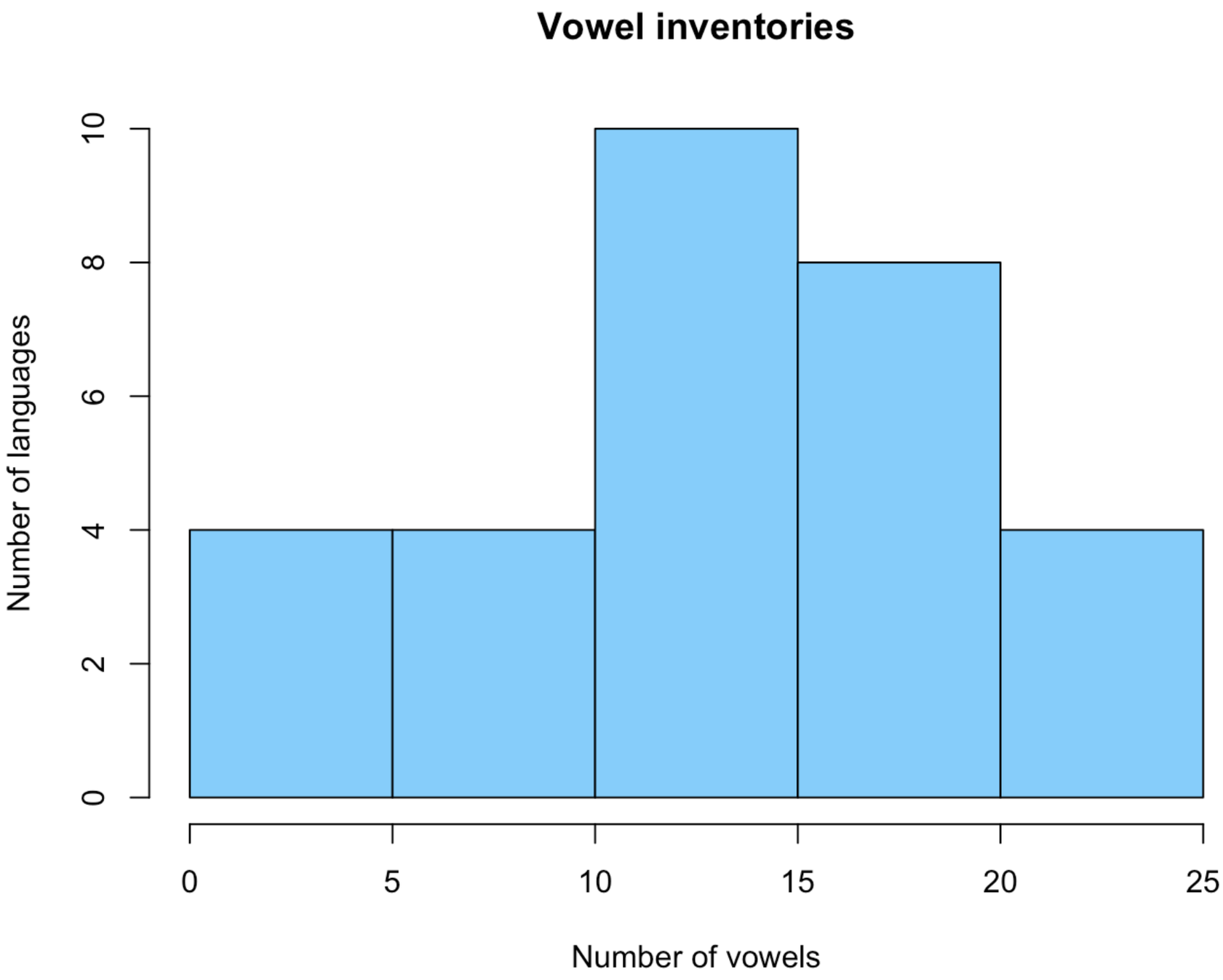
- `col = "red"` changes the color of the histogram

- you can get a list of other colors by calling the function `colors()`

- other optional arguments are:
  - main = 'Vowel inventories' inserts a title
  - xlab = '…' changes the name of the x axis
  - ylab = '…'

- use `?hist` to find out what other arguments it can take

Task: histogram



Installing packages

- R contains hundreds of packages, only some of them are preinstalled, most of them have to be installed individually.

- Two ways:

1. **Tools -> Install packages…** and then type the name of the package

2.
```
> install.packages("maps")
```
```
> install.packages("mapdata")
```

Loading packages

- To activate a package for the current R session, you need to load it:

```
> library(maps)
```

```
> library(mapdata)
```

Making your first map

- you can make use of the function `map()` and enjoy the map of the world:

```
> map()
```

- or restrict it to a particular region by limiting the axes:

```
> map(xlim = c(-20,40), ylim = c(40,60))
```

Making your first map

unction `map()` and enjoy

```
> map()
```

- or restrict it to a particular region by limiting the axes:

```
> map(xlim = c(-20,40), ylim = c(40,60))
```

Making your first map

unction `map()` and enjoy

```
```

- or restrict it to a particular region by limiting the axes:

```
> map(xlim = c(-20,40), ylim = c(40,60))
```

```
> map(xlim = c(20,180), ylim = c(30,90))
```

Making your first map

- or restrict it to a particular region by limiting the axes:

```
> map(xlim = c(-20,40), ylim = c(40,60))
```

```
> map(xlim = c(20,180), ylim = c(30,90))
```

Making your first map

- or select a country by typing ‘worldHires' and then the name of the country:

```
> map('worldHires','Germany')
```

Plotting points

- now that we have a map of Germany, we can add a point for our current location to it!

- for this you need to use the function `points()` and specify two arguments: longitude and latitude

```
> points(x = _ _ . _ _, y = _ _ . _ _)
```

longitude

latitude

Hint: longitude: 11.580738, latitude: 50.918549

Plotting points

- now that we have a map of Germany, we can add a point for our current location to it!

- for this you need to use the function specify two arguments: longitude a

```
> points(x = _ _ . _ _, y = _ _ .
```

longitude latitu

Hint: longitude: 11.580738, latitude: 50.9

Plotting points

- you can make your point more prominent by specifying its colour, size and shape:

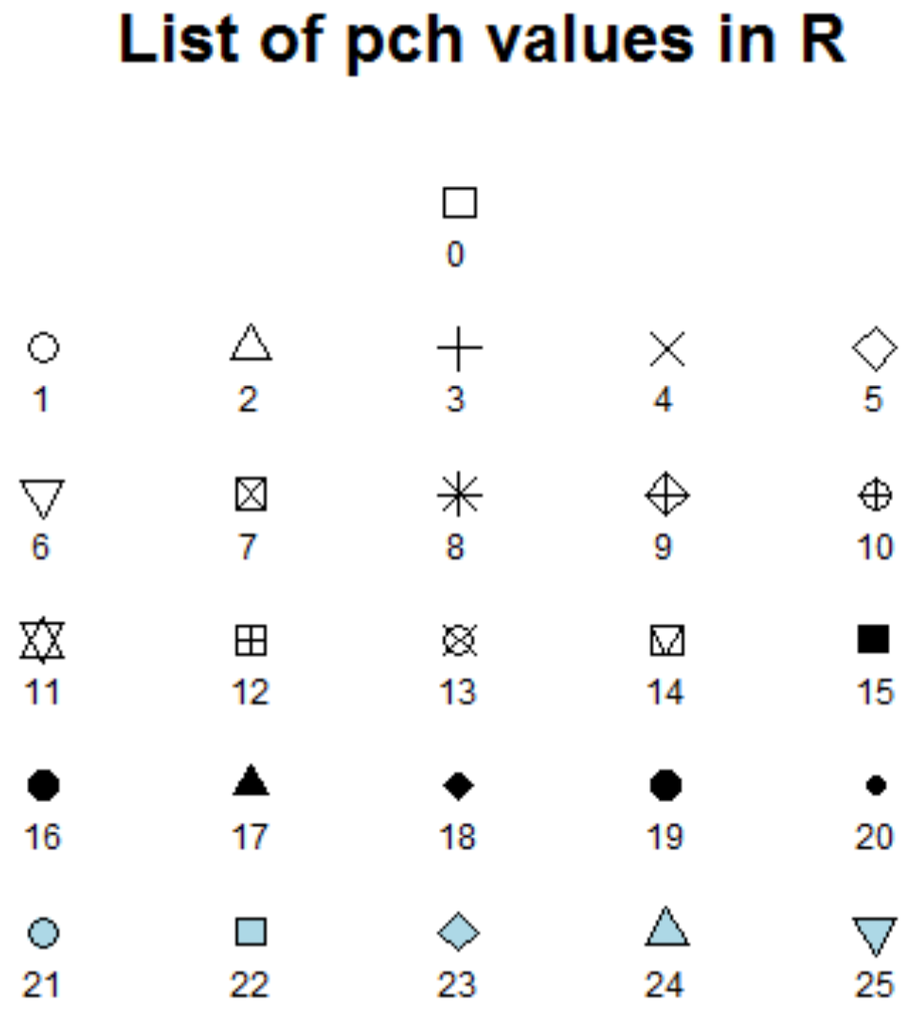
col | cex | pch = “plotting character”

- let’s add some power to our institute:

```
> points(11.580738, 50.918549,
pch = 20, cex=3, col="tomato")
```

(for some symbols, 21-25, you need to specify the background, e.g. `bg = "pink"`)

- go ahead and give the institute a new shape and colour!



Plotting points

- you can make your point more prominent by specifying its colour, size and shape:

col cex pch = “plotting character”

- let’s add some power to our institute:

```
> points(11.580738, 50.918549,
pch = 20, cex=3, col="tomato")
```

(for some symbols, 21-25, you need to specify the background, e.g. `bg = "pink"`)

- go ahead and give the institute a new shape and colour!

Layers

- making a map is done by putting layers on top of each other: the first layer is an empty map:

```
> map(xlim = c(20, 180), ylim = c(30, 90))
```

- points are plotted on top of it:

```
> points(124.0, 48.0, col = 'red', pch = 20)
> points(106.289, 48.324, col = 'red', pch = 20)
```

- the first point does not disappear, instead, the second point is added to the map

Plotting points

- But more often you need to plot more than one point!
- Remember how to put in several data points into one argument? Go ahead and add two points in one go.

Plotting points

- But more often you need to plot more than one point!

- Remember how to put in several data points into one argument? Go ahead and add two points in one go.

```
> points( c(124.0,106.289), c(48.0,48.324),
          col = 'red', pch = 20, cex = 3)
```

Plotting points

- But more often you need t
  point!

- Remember how to put in s
  one argument? Go ahead
  one go.

```
> points( c(124.0,106.289), c(48.0,48.324),
          col = 'red', pch = 20, cex = 3)
```

Plotting points

- But more often you need t
point!

- Remember how to put in s
one argument? Go ahead
one go.

```
> points( c(124.0,106.289), c(48.0,48.324),
          col = 'red', pch = 20, cex = 3)
```

- We will download a dataset from WALS, but before we continue, let's catch up on data formats

Data formats

a column in excel —> a ; or \t in a text editor

	A	B	C	D	E
1				Japanese	
2		**Meaning**	**Root**	**Standard romanization**	**IPA**
3					
4	1	**fire (n.)**	*pɨ(C)i	hi	çi
5			*u-matu		
6	2	**nose (n.)**	*pana	hana	hana
7	3	**go (v.)**	*ik-	ik-	ik-
8			*par-		
9	4	**water (n.)**	*mi-ntu	mizu	mizɯ
10	5	**mouth (n.)**	*kuti	kuti	kɯtɕi
11	6	**tongue (n.)**	*sita	sita	ɕita

- .csv **c**omma-**s**eparated **v**alues (; or ,)

```
;Meaning;Root;Japanese;
;;;Standard romanization;IPA
1;fire (n.);*p_(C)i  ;hi;◆i
;;*u-matu;;
2;nose (n.);*pana;hana;hana;
3;go (v.);*ik-;ik-;ik-
;;*par-;;
4;water (n.);*mi-ntu;mizu;miz_;
5;mouth (n.);*kuti;kuti;k_t_i
6;tongue (n.);*sita;sita;_ita
```

- .tsv **t**ab-**s**eparated **v**alues (\t)

Data formats

a column in excel —> a **;** or **\t** in a text editor

	A	B	C	D	E
1		**Meaning**	**Root**	**Japanese**	
2				**Standard romanization**	**IPA**
3					
4	1	**fire (n.)**	*pɨ(C)i	hi	çi
5			*u-matu		
6	2	**nose (n.)**	*pana	hana	hana
7	3	**go (v.)**	*ik-	ik-	ik-
8			*par-		
9	4	**water (n.)**	*mi-ntu	mizu	mizɯ
10	5	**mouth (n.)**	*kuti	kuti	kɯtɕi
11	6	**tongue (n.)**	*sita	sita	ɕita

- .csv **c**omma-**s**eparated **v**alues (**;** or **,**)

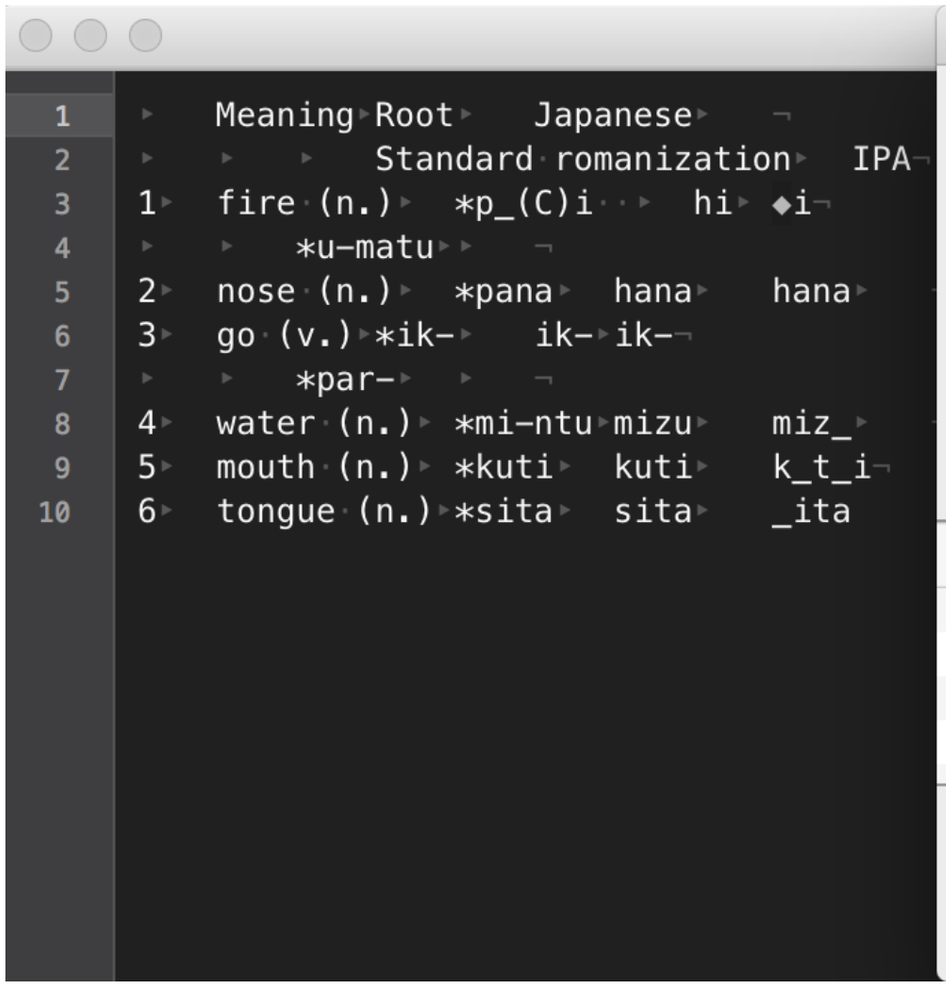
```
;Meaning;Root;Japanese;
;;;Standard romanization;IPA
1;fire (n.);*p_(C)i  ;hi;◆i
;;*u-matu;;
2;nose (n.);*pana;hana;hana;
3;go (v.);*ik-;ik-;ik-
;;*par-;;
4;water (n.);*mi-ntu;mizu;miz_;
5;mouth (n.);*kuti;kuti;k_t_i
6;tongue (n.);*sita;sita;_ita
```

- .tsv **t**ab-**s**eparated **v**alues (**\t**)

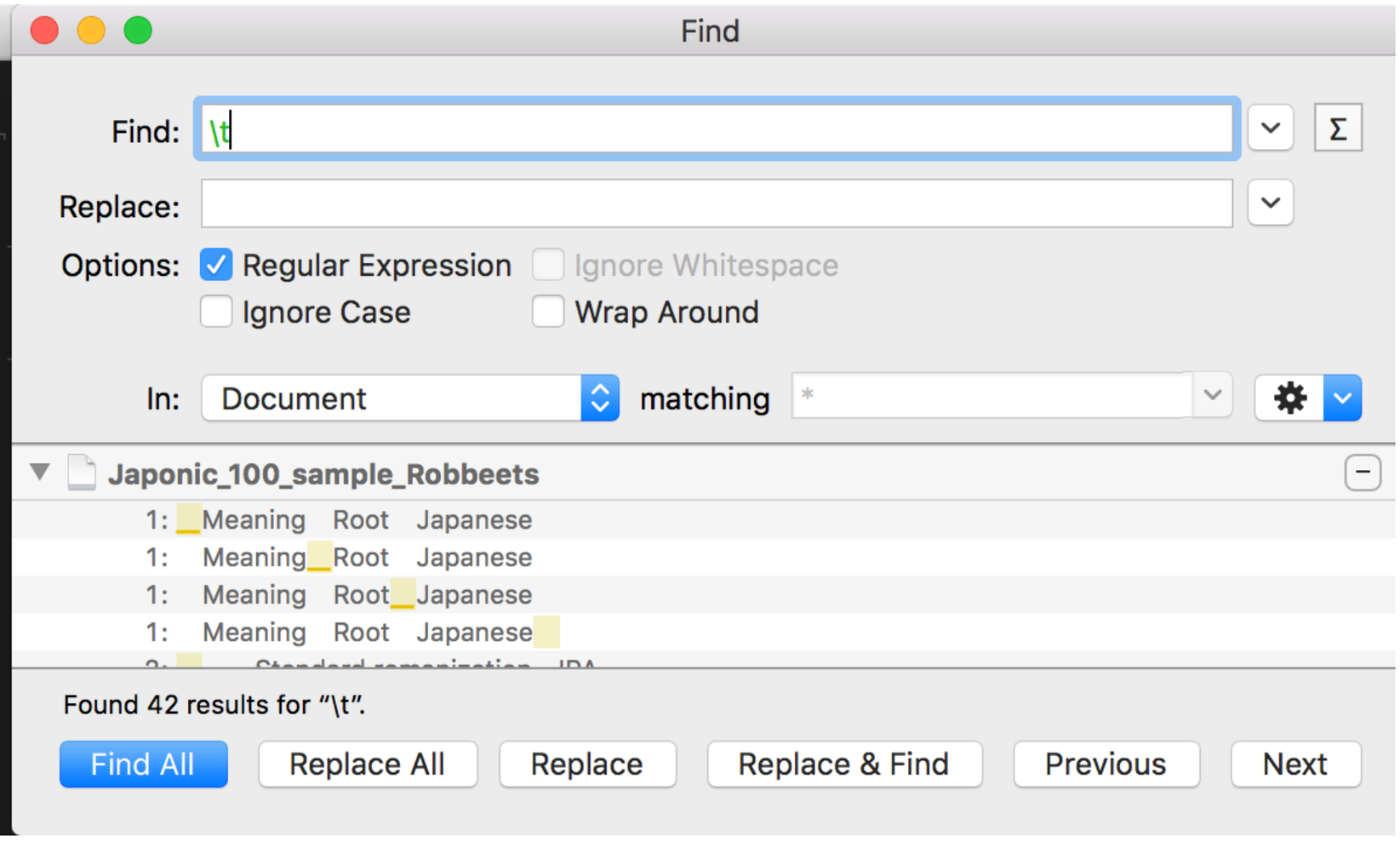
```
    Meaning Root    Japanese
                Standard romanization   IPA
1   fire (n.)   *p_(C)i    hi  ◆i
        *u-matu
2   nose (n.)   *pana   hana    hana
3   go (v.) *ik-    ik- ik-
        *par-
4   water (n.)  *mi-ntu mizu    miz_
5   mouth (n.)  *kuti   kuti    k_t_i
6   tongue (n.) *sita   sita    _ita
```

What's with this \t?

- **\t** stands for the 'tab' and is a regular expression



```
1   ▸  Meaning▸Root▸  Japanese▸  ¬
2   ▸  ▸  ▸  Standard·romanization▸  IPA¬
3   1▸ fire·(n.)▸  *p_(C)i··▸  hi▸ ◆i¬
4   ▸  ▸  *u-matu▸▸  ¬
5   2▸ nose·(n.)▸  *pana▸  hana▸  hana▸
6   3▸ go·(v.)▸*ik-▸  ik-▸ik-¬
7   ▸  ▸  *par-▸  ▸  ¬
8   4▸ water·(n.)▸ *mi-ntu▸mizu▸  miz_▸
9   5▸ mouth·(n.)▸ *kuti▸  kuti▸  k_t_i¬
10  6▸ tongue·(n.)▸*sita▸  sita▸  _ita
```



Other regular expressions would be **\n** (new line), **\d** (a digit), **\D** (a non-digit), etc.

You can find an interactive fun course on regular expressions here.

Getting the data

- Let us read in some languages with their coordinates from WALS, Feature 117A: Predicative Possession. First open the data as **tab-separated values**, copy and paste it into a new text editor file

Cleaning the data

- Now delete the head of the file:

```
Leon Stassen. 2013. Predicative Possession.
In: Dryer, Matthew S. & Haspelmath, Martin (eds.)
The World Atlas of Language Structures Online.
Leipzig: Max Planck Institute for Evolutionary Anthropology.
(Available online at http://wals.info/chapter/117, Accessed on 2017-10-30.)


abk Abkhaz 5 'Have' 43.0833333333 41.0 Northwest Caucasian Northwest Caucasian Simple
ace Acehnese 3 Topic 5.5 95.5 Malayo-Sumbawan Austronesian Simple Clauses
acl Acholi 4 Conjunctional 3.0 32.6666666667 Nilotic Eastern Sudanic Simple Clauses
ahu Aghu 3 Topic -6.16666666667 140.166666667 Awju-Dumut Trans-New Guinea Simple
```

(You could also open this text file in excel by clicking **Open with >** and see how it is represented there.)

Reading in the data

read the file into R:

```
> possession <- read.csv(file.choose(), sep = "\t")
```

or

```
> possession <- read.csv("filename.txt", sep = "\t")
```

:

the name of your file goes here

but remember to check and set your working directory before doing so:

```
> getwd()
> setwd("/bla/bla")
```

Inspecting the data

now have a look at what the first rows of the data look like:

```
> head(possession)
```

you might get something like this:

```
  wals.code      name value   description  latitude longitude                genus
1       abk    Abkhaz     5        'Have' 43.083333  41.00000 Northwest Caucasian
2       ace Acehnese     3         Topic  5.500000  95.50000     Malayo-Sumbawan
3       acl   Acholi     4 Conjunctional  3.000000  32.66667             Nilotic
4       ahu     Aghu     3         Topic -6.166667 140.16667         Awju-Dumut
5       agl    Aghul     1    Locational 41.750000  47.66667             Lezgic
6       ain     Ainu     5        'Have' 43.000000 143.00000               Ainu
                family            area
1 Northwest Caucasian Simple Clauses
2       Austronesian Simple Clauses
3    Eastern Sudanic Simple Clauses
```

Inspecting the data

now have a look at what the first rows of the data look like:

```
> head(possession)
```

you might get something like this:

	wals.code	name	value	description	latitude	longitude	genus
1	abk	Abkhaz	5	'Have'	43.083333	41.00000	Northwest Caucasian
2	ace	Acehnese	3	Topic	5.500000	95.50000	Malayo-Sumbawan
3	acl	Acholi	4	Conjunctional	3.000000	32.66667	Nilotic
4	ahu	Aghu	3	Topic	-6.166667	140.16667	Awju-Dumut
5	agl	Aghul	1	Locational	41.750000	47.66667	Lezgic
6	ain	Ainu	5	'Have'	43.000000	143.00000	Ainu

	family	area
1	Northwest Caucasian	Simple Clauses
2	Austronesian	Simple Clauses
3	Eastern Sudanic	Simple Clauses

Inspecting the data

now have a look at what the first rows of the data look like:

```
> head(possession)
```

you might get something like this:

```
  wals.code      name value   description  latitude longitude                genus
1       abk    Abkhaz     5        'Have' 43.083333  41.00000 Northwest Caucasian
2       ace Acehnese     3         Topic  5.500000  95.50000     Malayo-Sumbawan
3       acl   Acholi     4 Conjunctional  3.000000  32.66667             Nilotic
4       ahu     Aghu     3         Topic -6.166667 140.16667         Awju-Dumut
5       agl    Aghul     1    Locational 41.750000  47.66667             Lezgic
6       ain     Ainu     5        'Have' 43.000000 143.00000              Ainu
                family            area
1 Northwest Caucasian Simple Clauses
2       Austronesian Simple Clauses
3    Eastern Sudanic Simple Clauses
```

how do we access a column in R?

(two possibilities)

Plotting points on the map

- Remember we needed the coordinate values to plot the points?

```
> points(x = _ _ . _ _, y = _ _ . _ _)
```

longitude latitude

- Now we can feed R our two columns in the x and y arguments:

```
> map()
> points(x = possession$longitude,
         y = possession$latitude)
```

Customizing points

- It would be interesting to have a map displaying possession types by alternating colours. We leave the choice of colors to R first:

```
> points(x = possession$longitude,y = possession$latitude,
        col = possession$description, pch = 20)
```

Customizing points

- It would be interesting to have a map displaying possession types by alternating colours. We leave the choice of colors to R first:

```
> points(x = possession$longitude,y = possession$latitude,
        col = possession$description, pch = 20)
```

- We can also make a list of nice colours and make R use them for different possession types on the map. **But how do we find out how many colors we need?**

Customizing points

- It would be interesting to have a map displaying possession types by alternating colours. We leave the choice of colors to R first:

```
> points(x = possession$longitude,y = possession$latitude,
        col = possession$description, pch = 20)
```

- We can also make a list of nice colours and make R use them for different possession types on the map. **But how do we find out how many colors we need?**

```
> table(possession$description)
```

Customizing points

- It would be interesting to have a map displaying possession types by alternating colours. We leave the choice of colors to R first:

```
> points(x = possession$longitude,y = possession$latitude,
        col = possession$description, pch = 20)
```

- We can also make a list of nice colours and make R use them for different possession types on the map. **But how do we find out how many colors we need?**

```
> table(possession$description)
```

```
   'Have' Conjunctional      Genitive    Locational         Topic
       63            59            22            48            48
```

Barplot: fresh up

- let's revise the kind of a barplot we can already make:

```
> barplot(table(possession$description),
          col = c("limegreen", "violet", "dodgerblue", "yellow2",
          "firebrick1"),
          ylim = c(0, 70))
```

Barplot: fresh up

- let's revise the kind of a barplot we can already make:

```
> barplot(table(possession$description),
          col = c("limegreen", "violet", "dodgerblue", "yellow2",
          "firebrick1"),
          ylim = c(0, 70))
```

- although we do know that it's about predicative possession, the reader doesn't! Let's change it:

Barplot: fresh up

- let's revise the kind of a barplot we can already make:

```
> barplot(table(possession$description),
          col = c("limegreen", "violet", "dodgerblue", "yellow2",
          "firebrick1"),
          ylim = c(0, 70))
```

- although we do know that it's about predicative possession, the reader doesn't! Let's change it:

```
> barplot(table(possession$description),
          col = c("limegreen", "violet", "dodgerblue", "yellow2",
          "firebrick1"),
          ylim = c(0, 70),
          main = "Predicative possession cross-linguistically")
```

Barplot: fresh up

- let's revise the kind of a barplot we can already make:

```
> barplot(table(possession$description),
          col = c("limegreen", "violet", "dodgerblue", "yellow2",
          "firebrick1"),
          ylim = c(0, 70))
```

- although we do know that it's about predicative possession, the reader doesn't! Let's change it:

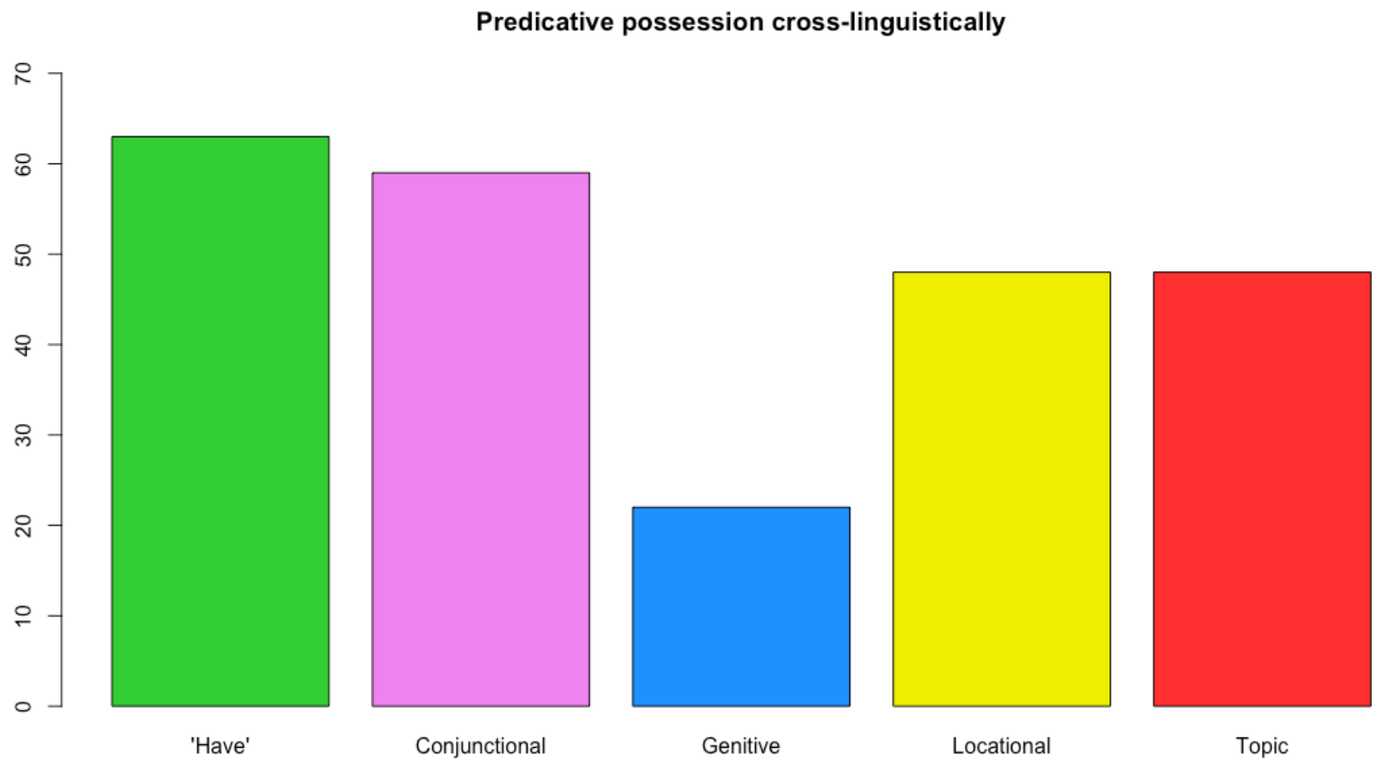
```
> barplot(table(possession$description),
          col = c("limegreen", "violet", "dodgerblue", "yellow2",
          "firebrick1"),
          ylim = c(0, 70),
          main = "Predicative possession cross-linguistically")
```

- alternatively you can specify the xlab and the ylab

Barplot: fresh up

- let's revise the kind of a

```
> barplot(table(possession$d
          col = c("limegreen
          "firebrick1"),
          ylim = c(0, 70))
```



- although we do know th
possession, the reader

```
> barplot(table(possession$description),
          col = c("limegreen", "violet", "dodgerblue", "yellow2",
          "firebrick1"),
          ylim = c(0, 70),
          main = "Predicative possession cross-linguistically")
```

- alternatively you can specify the xlab and the ylab

Customising points

- Save the colors you like in a vector:

```
> posscolors <- c("limegreen", "violet", "dodgerblue",
"yellow2", "firebrick1")
```

- condition the colors by the column on possession types:

```
> points(x = possession$longitude,
         y = possession$latitude,
         col = posscolors[possession$description])
```

vector with colors

column the colors depend on

Customising the basic map

Make profit of the optional arguments:

- no internal boundaries: `interior = FALSE`

(only if `fill = FALSE`, otherwise the argument is ignored)

- background in blue: `bg = "lightskyblue1"`
- continents in grey: `fill = TRUE, col = "lightgrey"`

(if `fill = FALSE`, the color is used for lines)

- add axes: `map.axes()`
- add a title: `title()`

e.g.

```
> map(bg="lightskyblue1", interior = FALSE)
> map(bg="lightskyblue1", fill=TRUE, col = "lightgrey")
> title("Predicative possession", cex=0.5)
```

Adding a legend and text

```
> legend("bottomright",
         legend = c("Have", "Conjunctional",
         "Genitive", "Locational", "Topic"),
          pch = 20,
         col = posscolors)
```

position

text of the legend

type of point

color vector

- add a WALS-code to each data point on the map

  (WARNING: looks ugly)

```
> text(x = possession$longitude,
       y = possession$latitude,
       possession$wals.code)
```

Working with data frames

- At times you have a list of questions as rows and a list of languages as column names. But this is not what you want as input for your nexus file!

- R offers a function `t()`, which transposes the data frame by reversing the rows and the columns

	Japanese			Miyako	Korean		Even
2	0	0	0	0	NA	0	0
3	0	0	0	0	NA	0	0
4	0	NA	NA	1	NA	1	0
5	0	0	0	1	NA	0	NA
6	0	0	0	1	1	1	0
7	0	0	0	0	NA	0	0
8	1	0	1	0	0	0	0
9	1	0	NA	1	NA	NA	0
10	0	0	1	1	0	1	1
11	1	0	1	1	NA	1	NA
12	1	0	1	1	1	NA	1
13	0	1	1	1	1	NA	1

```
1  "","V1","V2","V3","V4","V5","V6","V7
.  ","V27","V28","V29","V30","V31","V32
2  "Japanese",0,0,0,0,0,0,1,1,0,1,1,0,1
3  "Miyako",0,0,NA,0,0,0,0,0,0,0,0,1,0,
4  "Korean",0,0,NA,0,0,0,1,NA,1,1,1,1,1
5  "Even",0,0,1,1,1,0,0,1,1,1,1,1,1,0,1
6  "Evenki",0,0,NA,1,1,0,0,0,1,1,1,1,1,
7  "Manchu",NA,NA,NA,NA,1,NA,0,NA,0,NA,
8  "Negidal",0,0,1,0,1,0,0,NA,1,1,NA,NA
9  "Olcha",0,0,0,NA,0,0,0,0,1,NA,1,1,1,
```

Working with data frames

- download the data set *structural.csv* from github, folder *R course*

- read in your data in R:

```
> structural <- read.csv(file.choose(), sep = "\t")
```

- apply the transpose function `t()`:

```
> t_structural <- t(structural)
```

- save the result by applying the function `write.csv()`

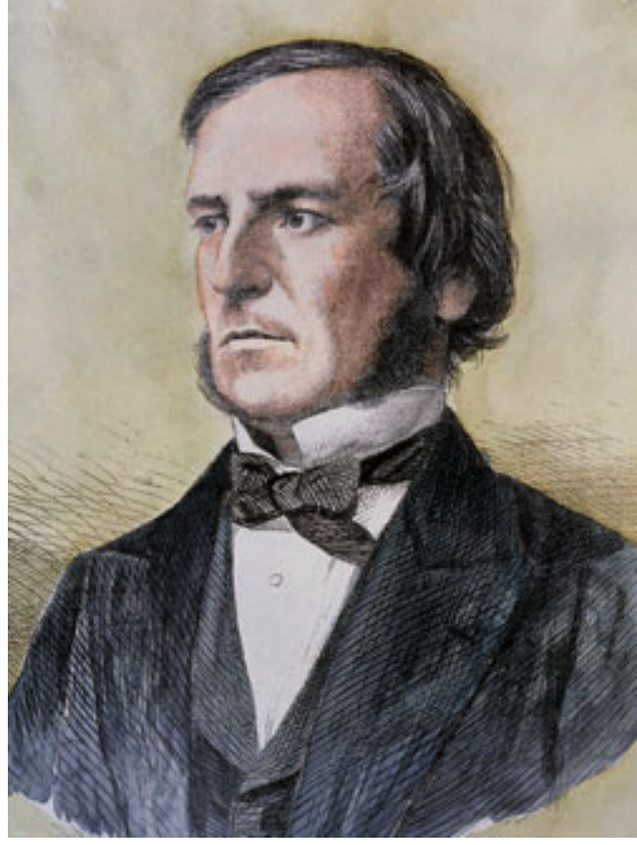
```
> write.csv(t_structural, "t_structural.txt")
```

Booleans

Boolean data type can take only two values: TRUE and FALSE.

Named after George Boole, who first defined an algebraic system of logic in the mid 19th century.

Wikipedia



Boolean operators:

> greater than

< less than

>= greater or equal to

<= less or equal to

== is equal to

!= is not equal to

- A Boolean condition evaluates to either TRUE or FALSE

- try typing some common sense thing in R console with a Boolean operator:

```
> 1 == 2
```

Booleans

Booleans

- let’s make a dummy vector n:

Booleans

- let's make a dummy vector n:

```
> n <- c(9,9,3,4,5)
```

Booleans

- let's make a dummy vector n:

```
> n <- c(9,9,3,4,5)
```

- hm, I wonder whether the first two numbers are the same

Booleans

- let's make a dummy vector n:

```
> n <- c(9,9,3,4,5)
```

- hm, I wonder whether the first two numbers are the same

```
> n[1] == n[2]
```

Booleans

- let's make a dummy vector n:

```
> n <- c(9,9,3,4,5)
```

- hm, I wonder whether the first two numbers are the same

```
> n[1] == n[2]
```

- check whether the number in **n[1]** is bigger than in **n[5]**

Booleans

- let's make a dummy vector n:

```
> n <- c(9,9,3,4,5)
```

- hm, I wonder whether the first two numbers are the same

```
> n[1] == n[2]
```

- check whether the number in **n[1]** is bigger than in **n[5]**

```
> n[1] > n[5]
```

Booleans

- download the data set called *phoible.csv* from github, folder *R course*

- read the dataset into R and calculate the number of languages, which have more than 20, 30, 50 consonants

Booleans

- download the data set called *phoible.csv* from github, folder *R course*

- read the dataset into R and calculate the number of languages, which have more than 20, 30, 50 consonants

```
> phoible <- read.csv(file.choose())
> sum(phoible$con>20)
```

Booleans

- download the data set called *phoible.csv* from github, folder *R course*

- read the dataset into R and calculate the number of languages, which have more than 20, 30, 50 consonants

```
> phoible <- read.csv(file.choose())
> sum(phoible$con>20)
```

- you can look inside the Boolean vector (warning: looks a bit scary):

Booleans

- download the data set called *phoible.csv* from github, folder *R course*

- read the dataset into R and calculate the number of languages, which have more than 20, 30, 50 consonants

```
> phoible <- read.csv(file.choose())
> sum(phoible$con>20)
```

- you can look inside the Boolean vector (warning: looks a bit scary):

```
> phoible$con>20
```

Booleans

- download the data set called *phoible.csv* from github, folder *R course*

- read the dataset into R and calculate the number of languages, which have more than 20, 30, 50 consonants

```
> phoible <- read.csv(file.choose())
> sum(phoible$con>20)
```

- you can look inside the Boolean vector (warning: looks a bit scary):

```
> phoible$con>20
```

- wonder what those languages are?

Booleans

- download the data set called *phoible.csv* from github, folder *R course*

- read the dataset into R and calculate the number of languages, which have more than 20, 30, 50 consonants

```
> phoible <- read.csv(file.choose())
> sum(phoible$con>20)
```

- you can look inside the Boolean vector (warning: looks a bit scary):

```
> phoible$con>20
```

- wonder what those languages are?

```
> phoible$language[phoible$con>50]
```

Booleans

Booleans

- you can play around with our previous dataset on predicative possession and **count the number of cases, for which the locational type “is true”**

Booleans

- you can play around with our previous dataset on predicative possession and **count the number of cases, for which the locational type "is true"**

```
> sum(possession$description=="Locational")
```

Conditional statements

An if-statement is a request to your computer to do a particular thing if a condition is met.

The if-statement consists of:

1. if
2. a condition in (), e.g. (n < m)
3. a trunk in {}

```
> n <- 1
> m <- 2
> if(n < m) {
  print("That's not surprising!") }
```

Conditional statements

An if-statement is a request to your computer to do a particular thing if a condition is met.

The if-statement consists of:

1. if
2. a condition in (), e.g. (n < m)
3. a trunk in {}

```
> n <- 1
> m <- 2
> if(n < m) {
  print("That's not
surprising!") }
```

we can express our astonishment about the number of consonants in some languages by:

Conditional statements

An if-statement is a request to your computer to do a particular thing if a condition is met.

The if-statement consists of:

1. if
2. a condition in (), e.g. (n < m)
3. a trunk in {}

```
> n <- 1
> m <- 2
> if(n < m) {
  print("That's not surprising!") }
```

we can express our astonishment about the number of consonants in some languages by:

```
> if (sum(phoible$con>50) > 10) {print ("Wow, that's a lot!")}
```

Loops

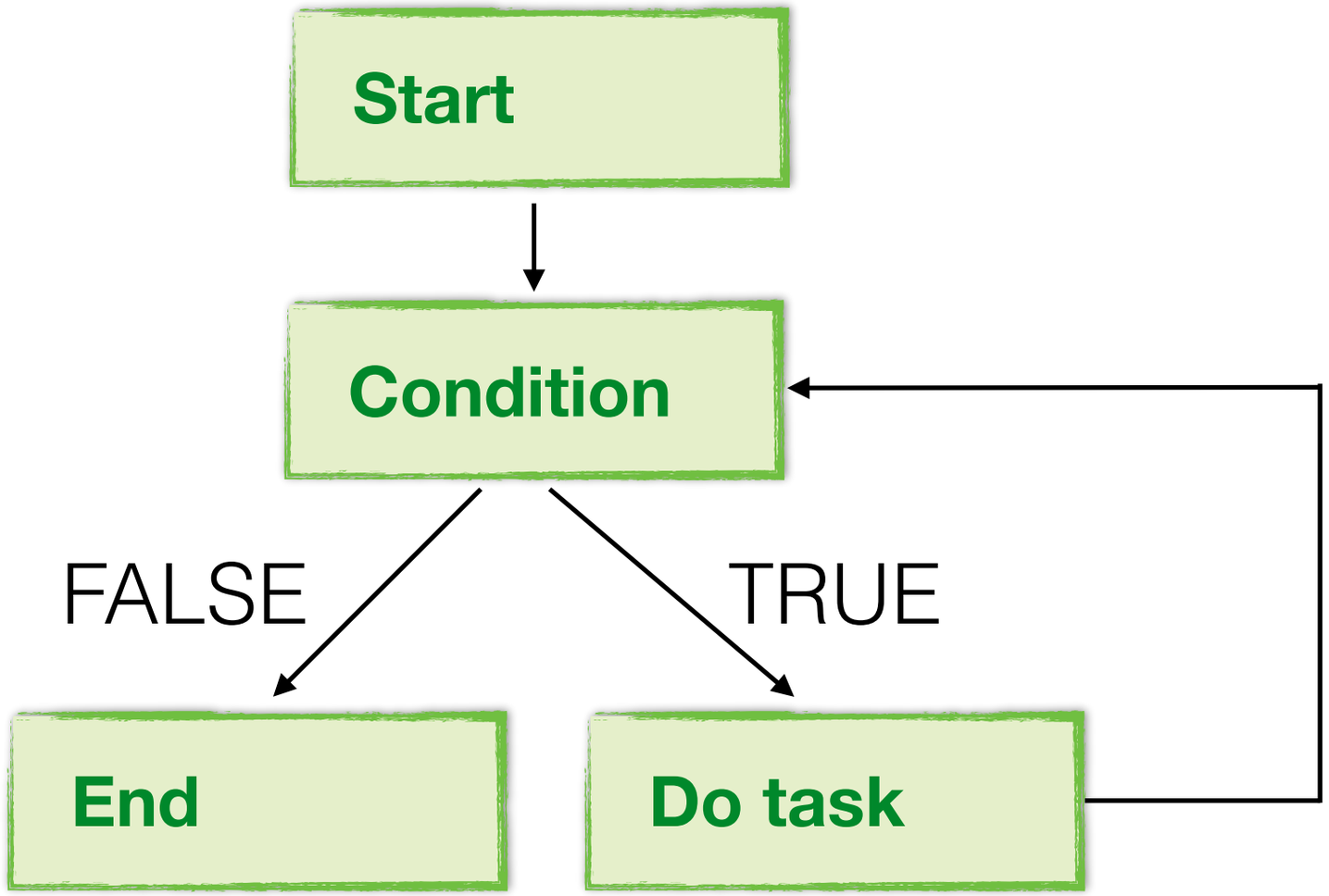
Loops

Two kinds of iterations:

1. **while:** do a particular thing as long as the condition is valid
2. **for:** do a particular thing as many times as I have decided

If you know exactly, how many times you want an operation to take place, for-loop is the right thing for you

Loops



for-loop

The for-loop consists of:

1. for
2. a count variable, e.g. i
3. a range, e.g. 1:5
4. a trunk in {}

```
> n <- c(1,2,3,4,5)

> for(i in 1:5) {
  n[i] <- n[i]*2
  print(n[i])
  }
```

for-loop

```
> n <- c(1,2,3,4,5)

> for(i in 1:5) {
print("We enter the loop with the value of n:")
print(n[i])
  n[i] <- n[i]*2
print("We leave the loop with the value of n:")
  print(n[i])
}

print(i)
```

```
> 1*2
> 2*2
> 3*2
> 4*2
> 5*2
```