Introduction to Programming in R

Ivan G. Costa & Tiago Maie
Institute for Computational Genomics
Joint Research Centre for Computational Biomedicine
RWTH Aachen University, Germany

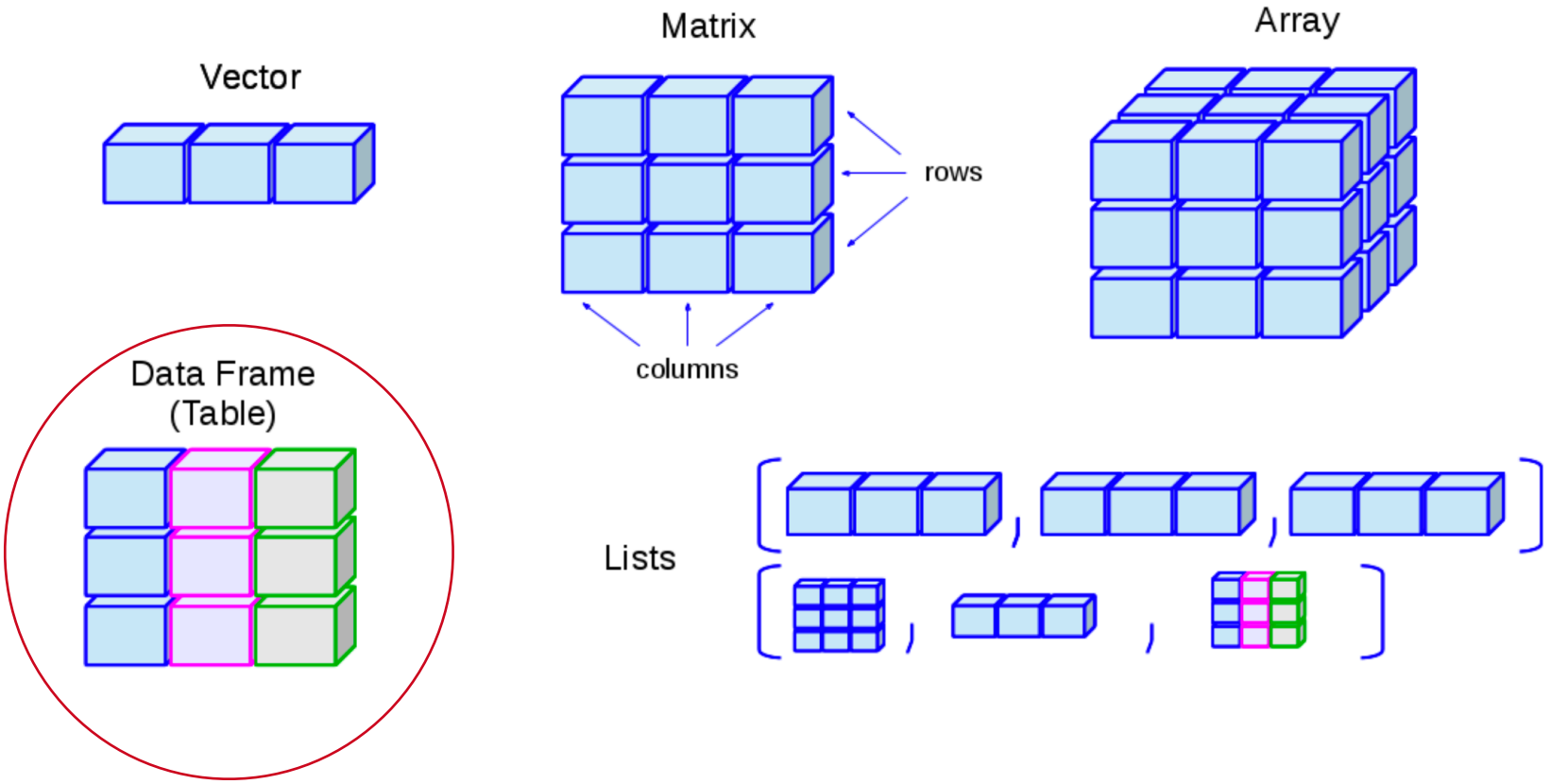
Institute for Computational Genomics
01011011010101
10100100101

RWTH AACHEN UNIVERSITY

Complex Data Structures

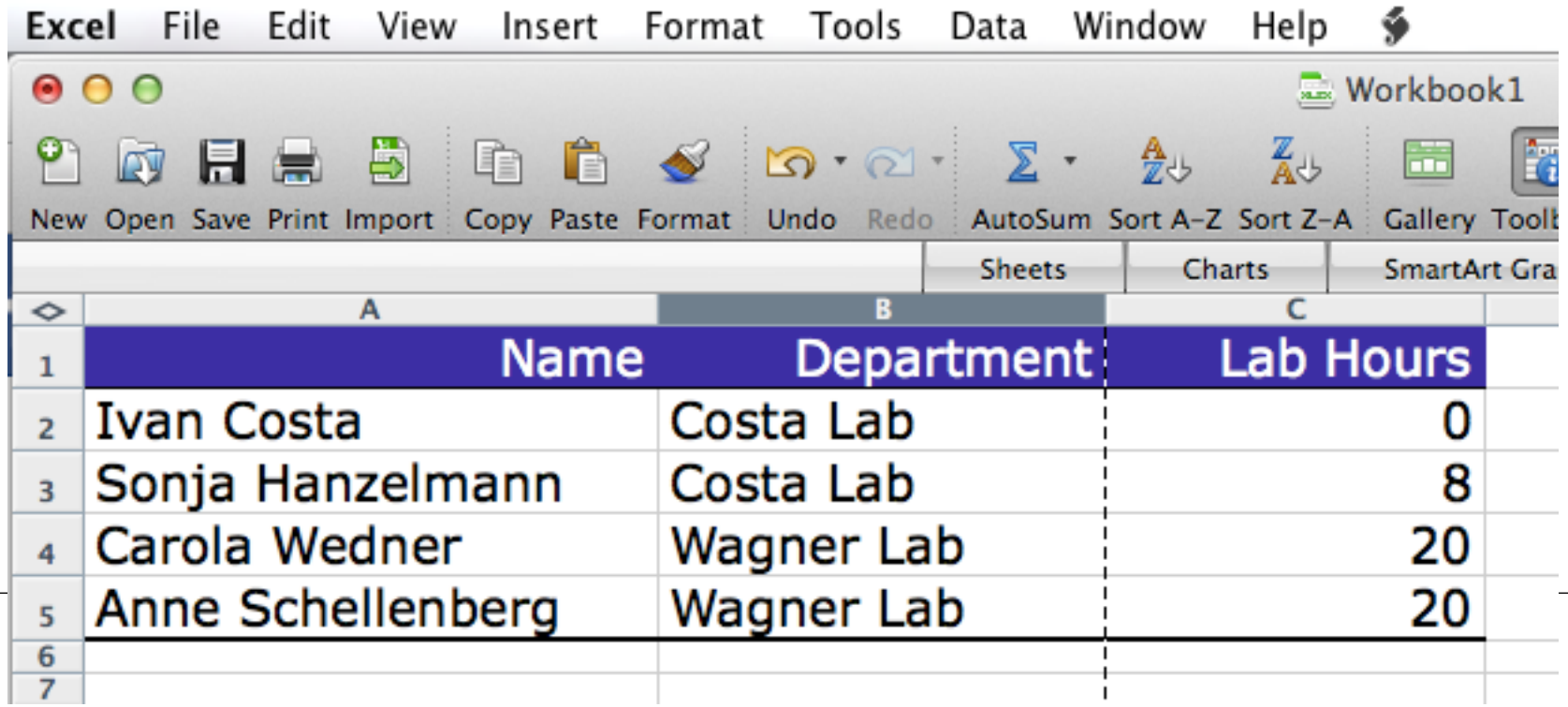
- Vector – variable containing an array of items of the same type
- Lists - a vector where items can have distinct types
- Matrix – two dimensional vector with items of the same type
- Data Frame – complex data structure for two dimensional data where columns can be of distinct type (as an excel sheet)

Complex Data Structures



Data Frames

- Data frames hold a spreadsheet like table. The observations are the rows and the covariates are the columns. Values in a column share the same type.
- Data frames can be operated as matrices and be indexed with two subscripts.



	A	B	C
1	Name	Department	Lab Hours
2	Ivan Costa	Costa Lab	0
3	Sonja Hanzelmann	Costa Lab	8
4	Carola Wedner	Wagner Lab	20
5	Anne Schellenberg	Wagner Lab	20
6			
7			

Data Frames

- Creation and manipulation

```
> data = data.frame(
      name = c("Ivan","Tiago","Carola","Anne"),
      department = c("Costa","Costa","Wagner","Wagner"),
      labhour = c(0, 8, 20, 20))
> data
    name department labhour
1   Ivan       Costa       0
2  Tiago       Costa       8
3 Carola      Wagner      20
4   Anne      Wagner      20
> data$department      # access department column of frame
[1] Costa  Costa  Wagner Wagner
Levels: Costa Wagner
> data[,"department"]  # access department column of frame
> data[,2]             # second column of data frame
```

Data Frames

- Creation and manipulation

```
> data[1,]                # first line of the data frame
name department labhour
1 Ivan       Costa        0
> rownames(data)      # row names
[1] "1" "2" "3" "4"
> rownames(data) = data$name
# make people names as row names
> data["Ivan",]         # find entries by first name
      name department labhour
Ivan     Ivan       Costa        0
```

Data Frames

- Creation and manipulation

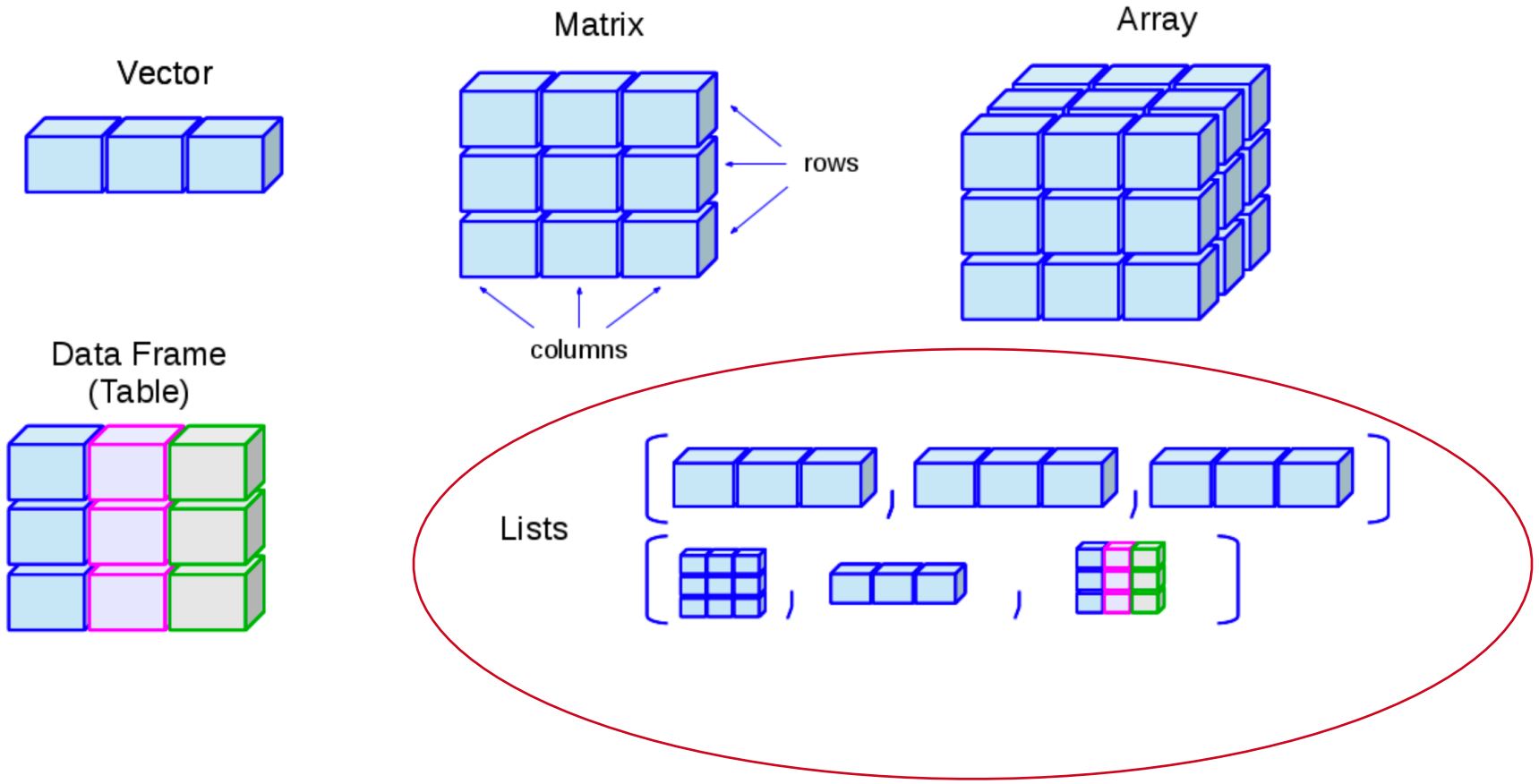
```
> data$labhour > 8     # lab hours exceeding 8
[1] FALSE FALSE  TRUE  TRUE
> data[data$labhour > 8,]
# data from members with more than 8 hours
    name department labhour
3 Carola      Wagner       20
4   Anne      Wagner       20
> data[data$department=="Costa",]
# data from members of Costa dept.
   name department labhour
1  Ivan       Costa        0
2 Tiago       Costa        8
```

Factor

- A list of categorical nature
  - i.e. gender (male, female, divers), department (wagner, costa), tumour type (…), cell type (…).
  - Important for statistical tests and plots

```
> data$department = as.factor(data$department)
> data$department
[1] Costa  Costa  Wagner Wagner
Levels: Costa Wagner
> levels(data$department)
[1] "Costa"  "Wagner"
> levels(data$department)=c("AG Costa","AG Wagner")
> data$department
> table(data$department)
  AG Costa  AG Wagner
```

Complex Data Structures



List

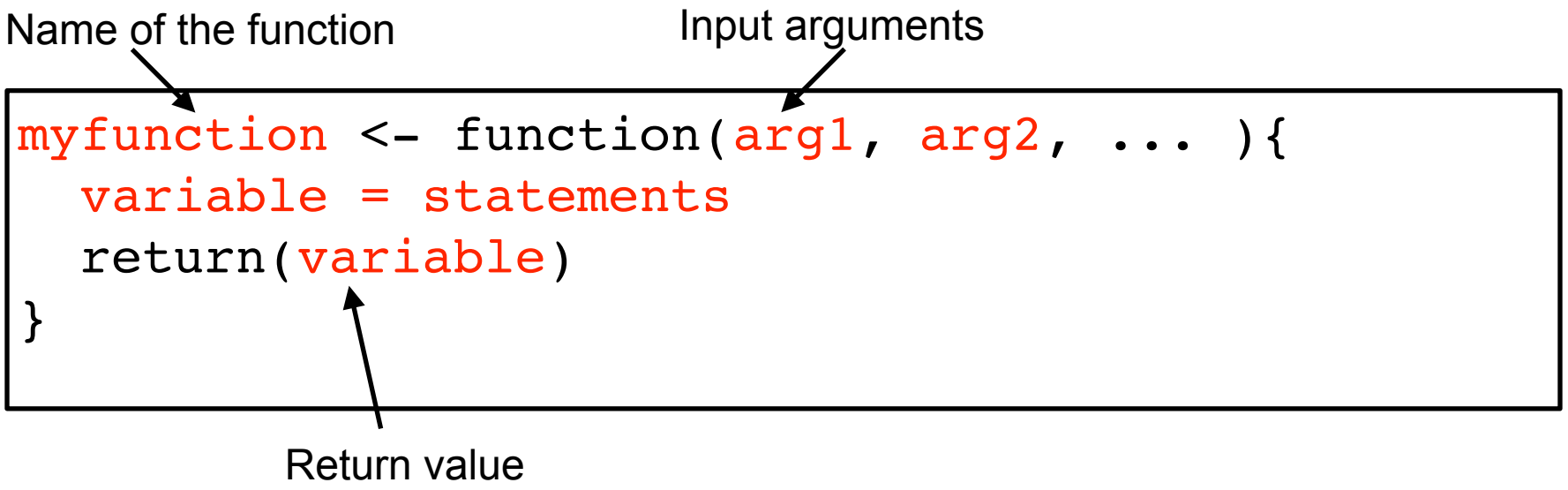
- An ordered collection of variables of distinct types under one variable (similar to a data frame for a single observation).

```
# example of a list with 3 components
> w = list(name="Fred", age=53, gender="male")
> w[[1]]       # access the first variable of the list
[1] "Fred"
> w$name       # access the variable "name" of the list
[1] "Fred"
> w[["age"]]   # access the variable age of the list
[1] 53
```

Own functions

Own Function

- Programming languages allow us to define our own functions. This is useful when you want to create a code describing a task that needs to be repeated (write a table as file, complex arithmetic calculation).



Own Function - Examples

```
myfunction <- function(arg1, arg2, ... ){
  variable = statements
  return(variable)
}
```

- Example of function for summing up 3 numbers

```
> sum3 <- function(a, b, c){
     # creates a function and stores in memory
     result = a + b + c;
     return(result)
   }
> sum3(3,4,5)
[1] 12
> sum3(1,2,3)
[1] 6
```

Exercises 1

1. Create a data frame with all members of your lab (or Class colleagues). Include information as age, gender, height (you can imagine this).

2. Create operations to list the name of all colleagues with age higher than 30.

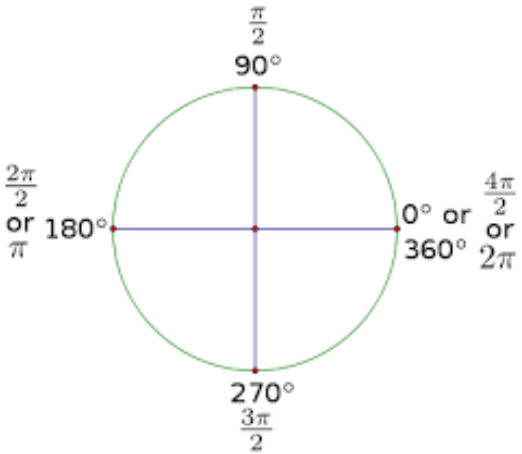
3. Update your method to list members with age higher than 30 and height higher than 170.

Exercise 2

- Create a function that receives 2 numbers and return their multiplication.
- Create a function that receives 4 numbers (or a vector of numbers) and returns a list with the minimum and the maximum values.

Exercise 3

- Create a function that takes degrees and returns radios. Use it to compute radian values for 90, 45 and 0 degrees (i.e. there is a variable **pi** in R)



$$\text{Radians} = \left(\frac{\pi}{180^{0}}\right) \times \text{degrees}$$

Exercises 4 (optional)

1. Use lists to redo the fruit shop exercise from the past day.

2. Can you tell the advantage of using lists instead of vectors?

Exercise 5 (optional)

- Create a function that converts Celsius to Fahrenheit degrees. Estimate the Fahrenheit for 40 or 0 degrees (Celsius).
- This is the conversion formula.

$$T_{(°\mathrm{F})} = T_{(°\mathrm{C})} \times 9/5 + 32$$

Control Commands

Control Commands

- Algorithms are usually not sequential.
- Control commands
  - test to decide the next steps
    - *if-else* command
  - Repeating commands until a condition is satisfied
    - *for* and *while*

if command

- only executed if condition is true

```
if (<logical test>){
   commands      # executed only if test is true
}
```

```
> grade = 6
> if (grade >= 6){
     print("fail")
  }
[1] "fail"
> grade = 4
> if (grade >= 6){
     print("fail")
  }
```

Algorithm Analysis

Algorithm Example - “Cake baking”

- Prepare a cake pan by spraying with baking spray or buttering and lightly flouring. Next, combine flour, baking powder, baking soda, and salt in a large bowl and set the mix aside. Add 3 eggs, one at a time, and mix just until combined. Add flour mixture and buttermilk, alternately, beginning and ending with flour. Preheat oven to 350º F, pour the dough in a pan and bake it for 25-30 minutes until edges turn loose from pan and toothpick inserted into middle of cake comes out clean. Remove from the oven and allow to cool for about 10 minutes.

Task - bake a cake

Language - English

Exact - ???

Well defined - ???

if-else command

```
if (<logical test>){
  commands  # executed only if test is true
}else{
  commands  # executed only if test is false
}
```

```
> grade = 4
> if (grade >= 6){
     print("fail")
  }else{
     print("pass")
  }
[1] "pass"
```

if-else function

- an if-else function variant that evaluates a vector of conditions

```
ifelse( vector conditions, expression 1, expression 2)
```

```
> ifelse(data$labhour > 8,"Biologist","Bioinformatician")
  [1] "Bioinformatician" "Bioinformatician"
  [3] "Biologist"        "Biologist"

> ifelse(data$labhour > 8, c(1,2,3,4), c(5,6,7,8))
[1] 5 6 3 4

# Note that expressions can also be lists
```

For command

- Repeat commands while interacting with a sequence of elements

```
for (value in sequence){
  commands # executed for every value in sequence
}
```

```
> lab_members = c("Ivan","Tiago","Carola","Anne")
> for (name in lab_members) {
     print(name)
  }
[1] "Ivan"
[1] "Tiago"
[1] "Carola"
[1] "Anne"
```

For command examples

```
> range = 1:6     # command that creates a vector from 1 to 6
> for (i in range) {
     print(i)
  }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
> value = 1
> for (i in range){
     value = value * i  # computes the factorial of 6
  }
> value
[1] 720
```

For command examples

```
> data = data.frame(name=c("Ivan","Sonja","Carola","Anne"),
   department=c("Costa","Costa","Wagner","Wagner"),
   labhour=c(0,8,20,20))
> rownames(data) = data$name # makes name the identifier for
each row.
> data["Ivan",]
> for (n in data$name){
     if (data[n,]$labhour > 8){
       print(data[n,]);
     }
  }
```

For command examples

```
> data = data.frame(name=c("Ivan","Sonja","Carola","Anne"),
   department=c("Costa","Costa","Wagner","Wagner"),
   labhour=c(0,8,20,20))
> rownames(data) = data$name # makes name the identifier for
each row.
> data["Ivan",]
> for (n in data$name){
     if (data[n,]$labhour > 8){
       print(data[n,]);
     }
  }
```

- What about previous example?

```
> data[data$labhour > 8,]
```

While command

- Repeats statement while condition is true

```
while (<logical test>){
  statements     # executed while test is true
}
```

```
> i = 1
> while (i < 6) {
      print(i)
      i = i+1
  }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

Exercises 6

1. Write a loop that prints numbers 4, 6, 8 and 10 to the screen.

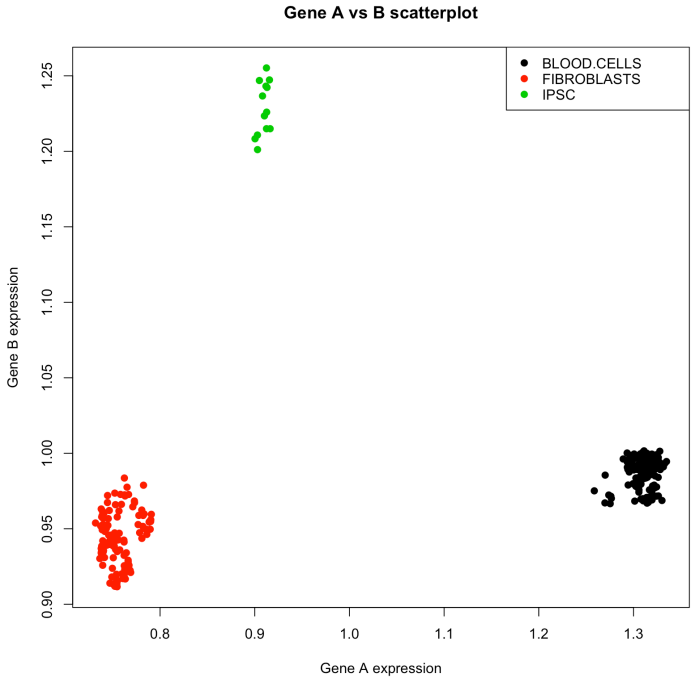
2. Write a loop that prints 1 to 10 and repeats this 3 times. Use a loop inside another loop to solve this.

3. Write a loop that writes all numbers from 1 to 35 but skips the numbers 3,9,13,19,23,29. Tips: you can use the operator **%in%** to check if a value is in a list and you need a loop and a if for this problem.

Plotting and Statistics / Afternoon

- R provides several functions for plotting and statistical analysis of data

- Example data
  - ~300 samples of blood, iPSC and fibroblast cells
  - 2 marker genes



- We will show how to perform scatter plots, pie charts, bar plots and statistical tests in this data
- Go to the handout!

Want more?

More training material:

https://rafalab.github.io/dsbook/r-basics.html#exercises-2

https://www.datamentor.io/r-programming/#tutorial

Inst. for Computational Genomics

- Ivan G. Costa
- Tiago Maie

Institute for Computational Genomics
01011011010101
10100100101

RWTH AACHEN UNIVERSITY