Universität Augsburg
Fakultät für Angewandte Informatik

Advanced Programming with R

1. Introduction to Programming

WS 2025/26

Dr. Cesar Ivan Alvarez

cesar.alvarez@uni-a.de

Agenda

1. What is Programming?
2. Programming with R
3. Programming Principles
4. Practice 1

Advance Programming with R

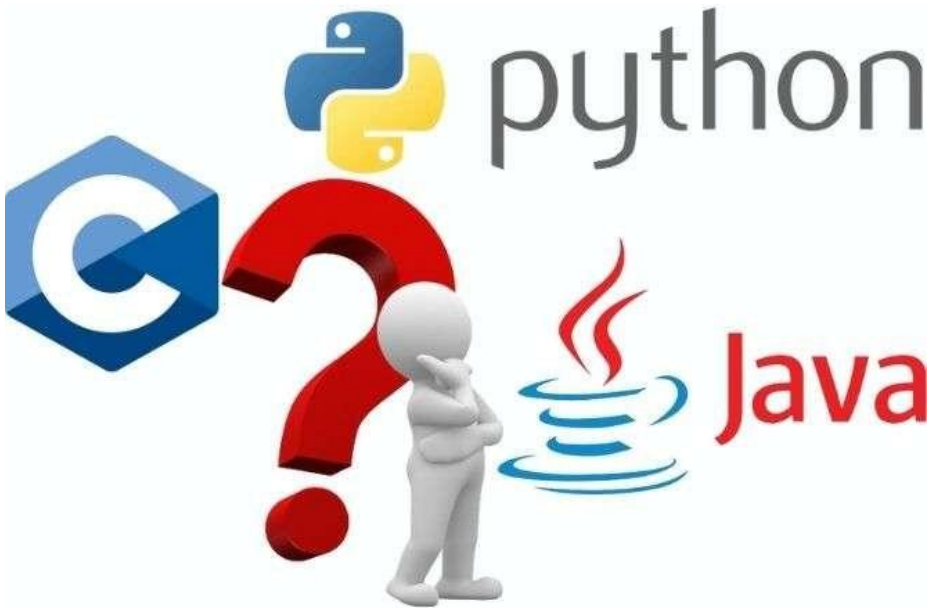
What is Programming?

- It is the process of writing precise and detailed instructions in a programming language so that a computer can perform a specific task. These instructions are called code, and they are organised logically so that the computer can understand and execute them.

Advance Programming with R

What is a programming language?

- A programming language is a tool that allows you to develop software or computer programmes. Programming languages are used to design and implement programmes responsible for defining and managing the behaviour of a computer's physical and logical devices.
- High-level programming languages allow commands to be given to the computer in a language similar to our own (Visual Basic, Pascal, Logo, C++, JavaScript, etc.) and always or almost always in English.



Advance Programming with R

The most used languages according to the TIOBE index, October 2025 (The TIOBE Programming Community index is an indicator of the popularity of programming languages).

Oct 2025	Oct 2024	Change	Programming Language	Ratings	Change
1	1		Python	24.45%	+2.55%
2	4	↑	C	9.29%	+0.91%
3	2	↓	C++	8.84%	-2.77%
4	3	↓	Java	8.35%	-2.15%
5	5		C#	6.94%	+1.32%
6	6		JavaScript	3.41%	-0.13%
7	7		Visual Basic	3.22%	+0.87%
8	8		Go	1.92%	-0.10%
9	10	↑	Delphi/Object Pascal	1.86%	+0.19%
10	11	↑	SQL	1.77%	+0.13%
11	9	↓	Fortran	1.70%	-0.10%
12	29	⇈	Perl	1.66%	+1.10%
13	17	⇈	R	1.52%	+0.43%
14	15	↑	PHP	1.38%	+0.17%

https://www.tiobe.com/tiobe-index/

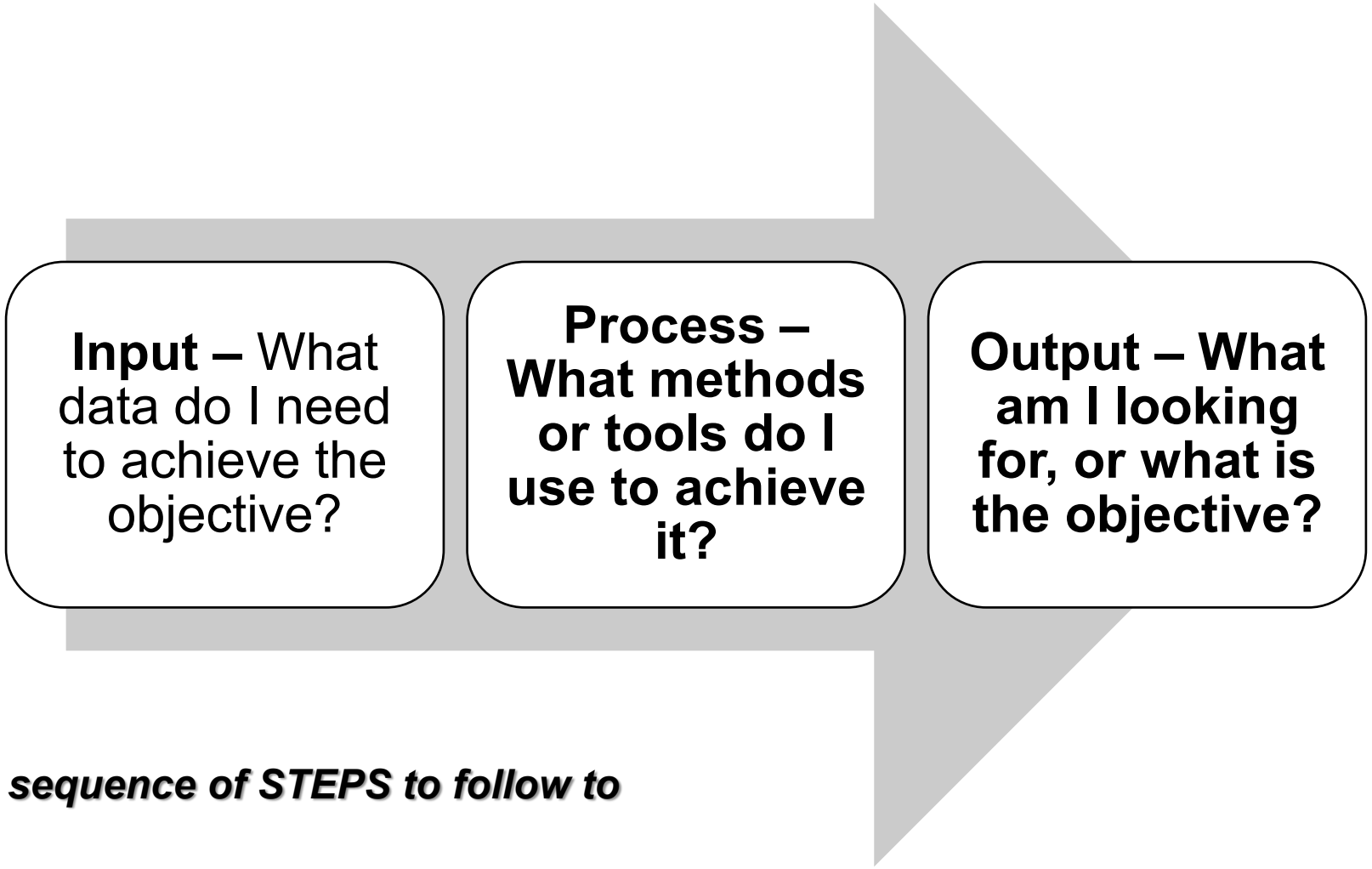
Advance Programming with R

What is R and R-Studio?

- **R** is a programming language and development environment designed specifically for statistical analysis and data manipulation. It is free and open-source software that provides a wide range of tools and libraries for data processing and graphics generation.
- **RStudio** is an integrated development environment (IDE) for the R programming language, dedicated to statistical computing and graphics. It includes a console, syntax editor that supports code execution, as well as tools for plotting, debugging, and workspace management.

Advance Programming with R

What is an algorithm?

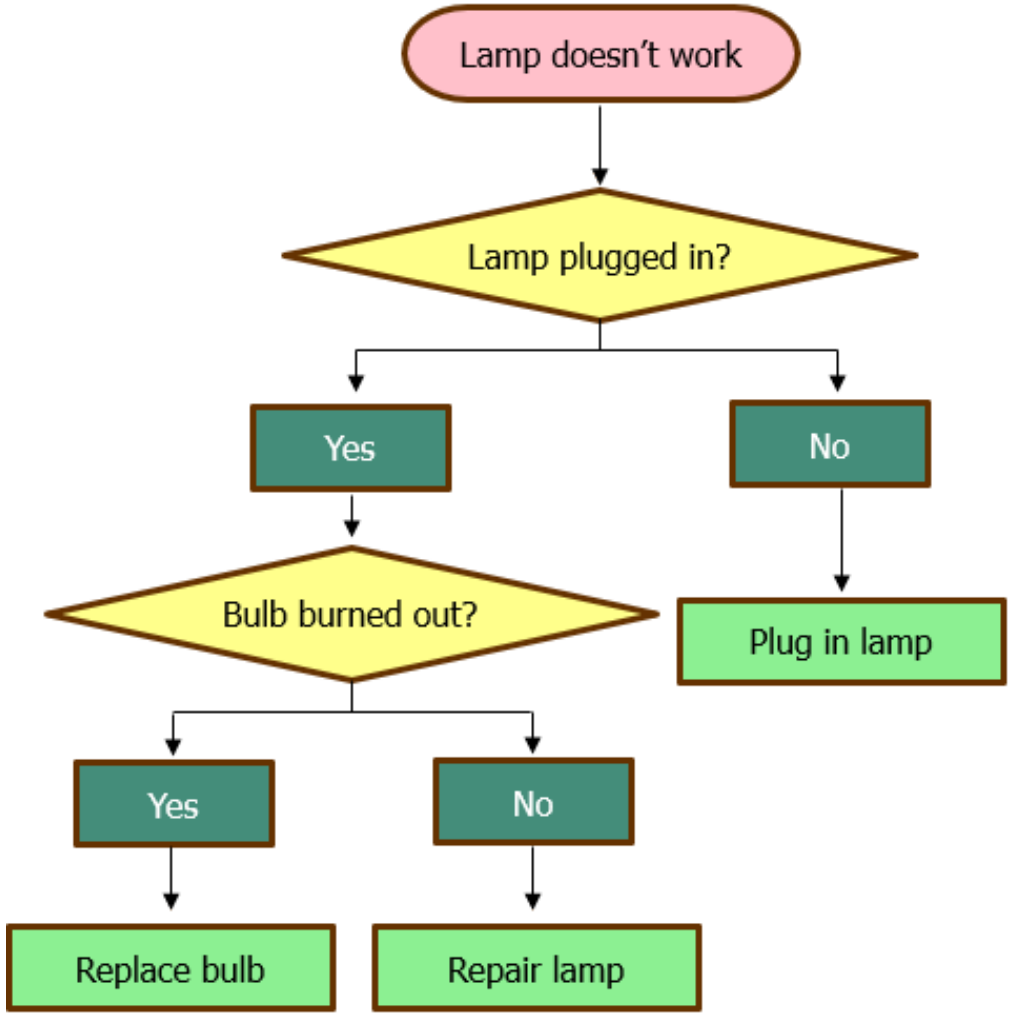


An algorithm is a sequence of STEPS to follow to solve a problem.

Advance Programming with R

What is a flowchart?

A flowchart is a graphical representation of the algorithm.



Programming Principles

Variables and Constants

- A variable is a named data element whose value can change during the course of a programme's execution. A variable name must follow the naming convention for an identifier (an alphabetic character or number and the underscore sign).

a <- 1
b <- 2
c <- a+b
print(c)
 	
nam = "Cesar“
cat(“Hi my name is", nam)
 	
decision = TRUE //Boolean
variable in R TRUE or FALSE
	a = 1
	b = 2
	a+b
	print(c)
	
	nam = 'Cesar'
	print(‘Hi my name is ' + nombre)
	
	decision = True #Boolean
	variable in Python True or False

Programming Principles

Variables and Constants

- There are other types of input variable structures such as matrices, dataframes, lists, dictionaries, among others.

dataset_r <- data.frame(
  ID = 1:5,
  Name = c("Alice", "Bob", "Charlie", "David", "Eve"),
  Age = c(25, 30, 22, 35, 28),
  Score = c(85.5, 90.0, 78.5, 88.0, 92.5))
dataset_r$Name
dataset_r[,3]
 	
dictionary_r <- list(
  name = "Alice",
  age = 25,
  scores = c(85, 90, 78))
print(dictionary_r$name)
	import pandas as pd
	dataset_py = pd.DataFrame({
	  'ID': [1, 2, 3, 4, 5],
	  'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
	  'Age': [25, 30, 22, 35, 28],
	  'Score': [85.5, 90.0, 78.5, 88.0, 92.5]})
	dataset_py["Name"]
	dataset_py.iloc[:, 2]
	dataset_py.loc[:, 'Age':'Score']
	dictionary_py = {
	  "name": "Alice",
	  "age": 25,
	  "scores": [85, 90, 78]}
	print(dictionary_py["name"])

Programming Principles

Conditionals

- Conditionals are structures that allow you to choose between executing one action or another. They are related to the conditional ‘if’ that we use in a sentence. In English, the words if and else are used.

```r
a <- 200
b <- 125

if (b > a) {
  c <- 2*a + b
  cat("b is greater than a and c is",c)
  } else if (a == b) {
  print("a and b are equal")
} else {
  c <- a + 2*b
  cat("a is greater than band c is",c)
}
```

```python
a = 100
b = 125

if b > a:
    c = 2 * a + b
    print(f"b is greater than a and c is {c}")
elif a == b:
    print("a and b are equal")
else:
    c = a + 2 * b
    print(f"a is greater than b and c is {c}")
```

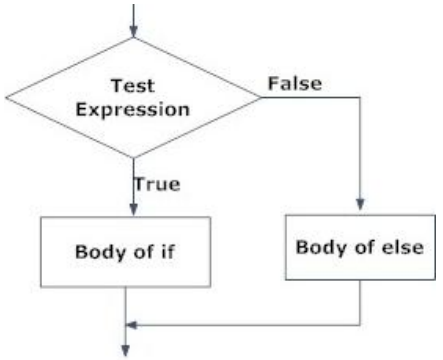


Fig: Operation of if...else statement

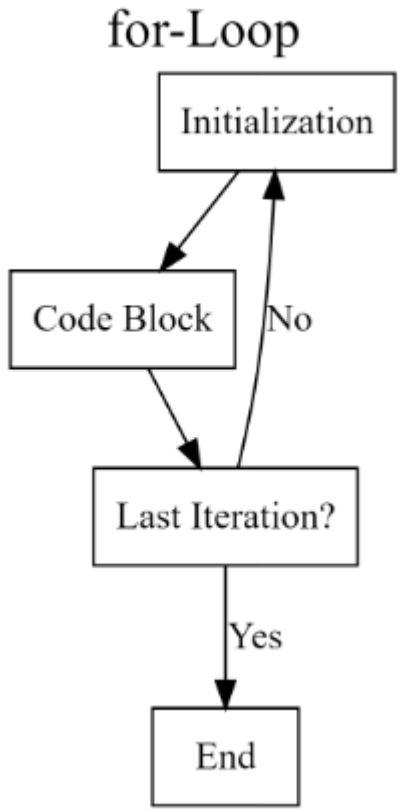
Programming Principles

Loops

- Loops are sequences of cyclical instructions that allow us to repeat an action while a certain condition is being met, and the process does not stop until this condition is no longer met.

for(number in 1:10) {
  print(number)
}
 	
v1 <- c(3,4,5)
v2 <- c(3,4,6)
v3 <- c(0,0,0)
n = length(v1)
 	
for (i in 1:n) {
  v3[i] <- v1[i] + v2[i]
}
V3
	for number in range(1, 11):
	  print(number)
	
	v1 = [3, 4, 5]
	v2 = [3, 4, 6]
	v3 = [0, 0, 0]
	n = len(v1)
	
	for i in range(n):
	  v3[i] = v1[i] + v2[i]
	
	print(v3)



Programming Principles

Loops

for (i in 1:10) {
if (i == 5) {
break
}
cat("The i-value is:", i, "\n")}
	for i in range(1, 11):
	if i == 5:
	break
	print("The i-value is:", i)

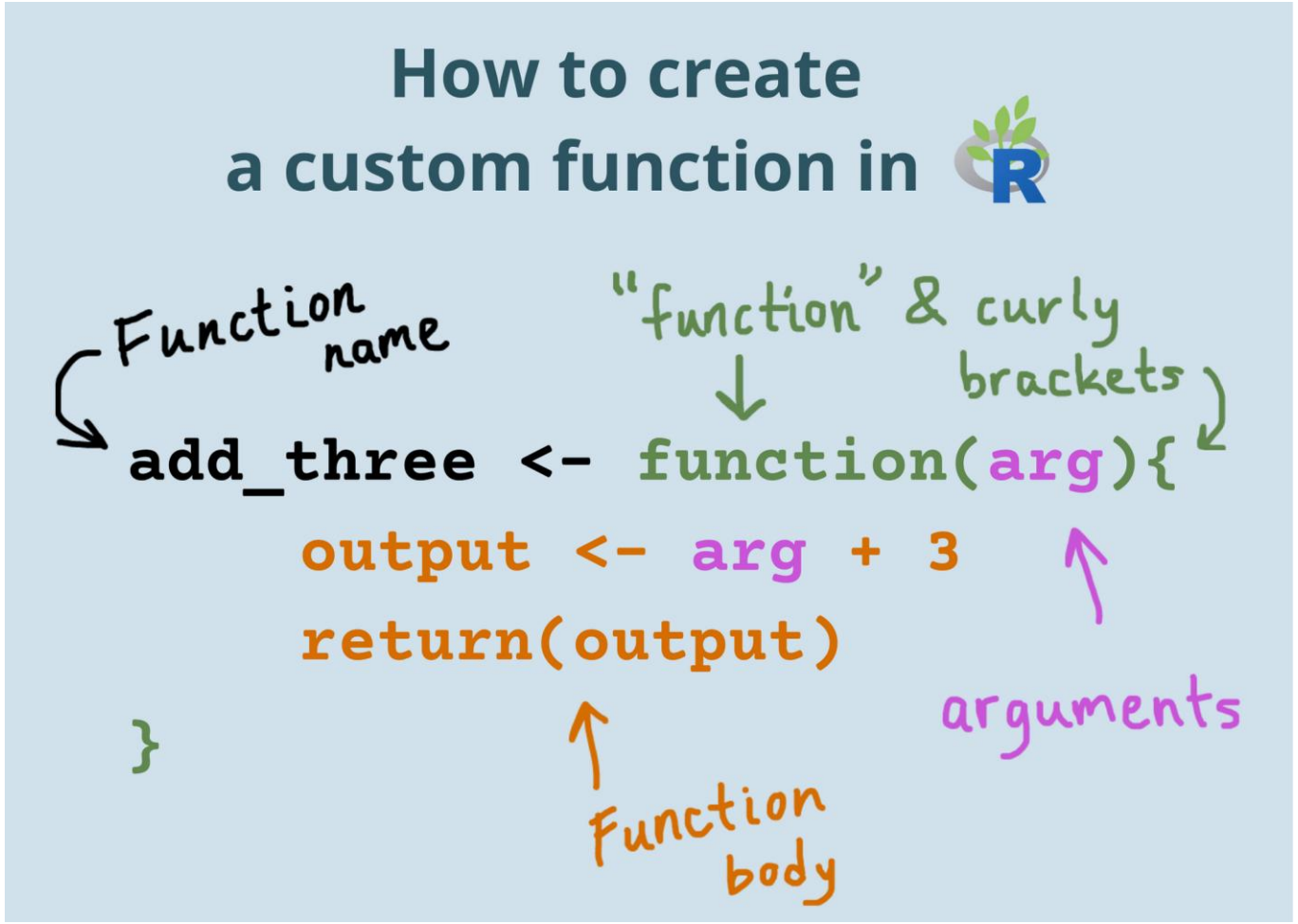
for (i in 1:5) {
if (i == 3) {
next
}
cat("The i-value is:", i, "\n")}
	for i in range(1, 6):
	if i == 3:
	continue
	print("The i-value is:", i)

- The break statement is used to exit a loop immediately when a condition is met.
- The next statement is used to skip the current iteration and continue with the next iteration of the loop, similar to the continue statement in Python.

Programming Principles

Functions

- A function in R is an object containing multiple interrelated statements that are run together in a predefined order every time the function is called. Functions in R can be built-in or created by the user (user-defined). The primary purpose of creating a user-defined function is to optimise our program by avoiding the repetition of code blocks used for frequently performed tasks in a particular project, preventing inevitable and hard-to-debug errors related to copy-paste operations, and enhancing code readability. A good practice is to create a function whenever we need to run a certain set of commands more than twice.



Programming Principles

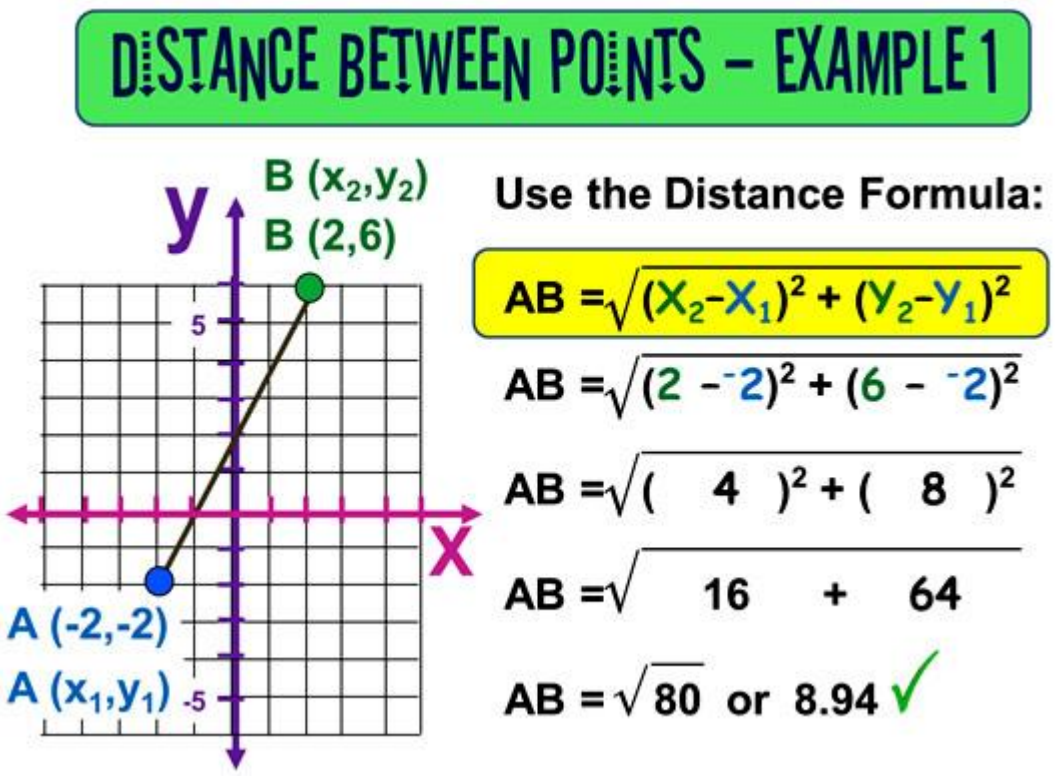
Functions

mean_two_numbers <-
function(num_1, num_2) {
  mean <- (num_1 + num_2) / 2
  return (mean)}
 	
mean_two_numbers(1,3)
	def mean_two_numbers(num_1,
	num_2):
	  mean = (num_1 + num_2) / 2
	  return mean
	
	mean_two_numbers(1, 3)

Programming Principles

Functions




```r
distance_2d <- function(x1, y1, x2, y2) {
  # Calculate the difference in x-coordinates squared
  delta_x_sq <- (x2 - x1)^2

  # Calculate the difference in y-coordinates squared
  delta_y_sq <- (y2 - y1)^2

  # Apply the Pythagorean theorem (square root of the sum)
  distance <- sqrt(delta_x_sq + delta_y_sq)

  return(distance)
}

distance_2d(-2,-2,2,6)
```

Practice 1

Loading Geospatial Data (Shapefiles)

```
# Check if the sf package is installed, and install it only if it's missing
if (!requireNamespace("sf", quietly = TRUE)) {
  install.packages("sf")}
library(sf) # Load the sf library
shapefile_path <- "C:/uoc/CNTR_RG_20M_2024_4326.shp/CNTR_RG_20M_2024_4326.shp" # Define the path to your shapefile https://ec.europa.eu/eurostat/web/gisco/geodata/administrative-units/countries
shapefile_data <- st_read(shapefile_path) # Read the shapefile
plot(shapefile_data$geometry) # Plot the shapefile
crs_info <- st_crs(shapefile_data) # Get CRS (Coordinate Reference System)
print(paste("CRS:", crs_info$epsg, "-", crs_info$proj4string))
fields <- colnames(shapefile_data) # Get field names (attribute columns)
print(fields)
```

Practice 1

Loading Geospatial Data (Shapefiles)

RStudio

File Edit Code View Plots Session Build Debug Profile Tools Help

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

Untitled1* | 01_CARGA_SHAPEFILES.R

Source on Save | Run | Source

```
1  # Check if the sf package is installed, and install it only if it's missing
2  if (!requireNamespace("sf", quietly = TRUE)) {
3    install.packages("sf")
4  }
5
6  # Load the sf library
7  library(sf)
8
9  # Define the path to your shapefile
10 shapefile_path <- "C:/Users/alvarece/Desktop/MODELAMIENTO Y PROGRAMACION GIS/PRACTIC
11
12 # Read the shapefile
13 shapefile_data <- st_read(shapefile_path)
14
15 # Print summary
16 print(shapefile_data)
17
18 # Plot the shapefile
19 plot(shapefile_data$geometry)
20
21 # Get CRS (Coordinate Reference System)
22 crs_info <- st_crs(shapefile_data)
23 print(paste("CRS:", crs_info$epsg, "-", crs_info$proj4string))
24
25 # Get field names (attribute columns)
26 fields <- colnames(shapefile_data)
27 print("Fields in the shapefile:")
28 print(fields)
29
```

17:1 (Top Level) R Script

Environment | History | Connections | Tutorial

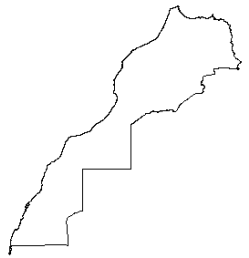
Import Dataset | 161 MiB | List

R | Global Environment

Data	
crs_info	List of 2
shapefile_data	1 obs. of 3 variables
Values	
fields	chr [1:3] "OBJECTID" "Shape__Are" "geome…
shapefile_path	"C:/Users/alvarece/Desktop/MODELAMIENTO …

Files | Plots | Packages | Help | Viewer | Presentation

Zoom | Export | Publish



Console | Terminal | Background Jobs

R 4.4.2 · ~/

```
> print("Fields in the shapefile:")
[1] "Fields in the shapefile:"
> print(fields)
[1] "OBJECTID"   "Shape__Are" "geometry"
>
```

Thank you for your attention

Universität Augsburg
Fakultät für Angewandte Informatik

Dr. Cesar Ivan Alvarez
Chair of Climate Resilience of Cultural Ecosystems
Universität Augsburg
cesar.alvarez@uni-a.de