university of groningen / faculty of economics and business / operations

# Data Analysis & Programming for Operations Management (DAPOM)

Wout van Wezel (Coordinator)
w.m.c.van.wezel@rug.nl

Jan Eise Fokkema
Onur Kilic
Jarno Kuik

(Nick Szirbik, Nicky van Foreest)

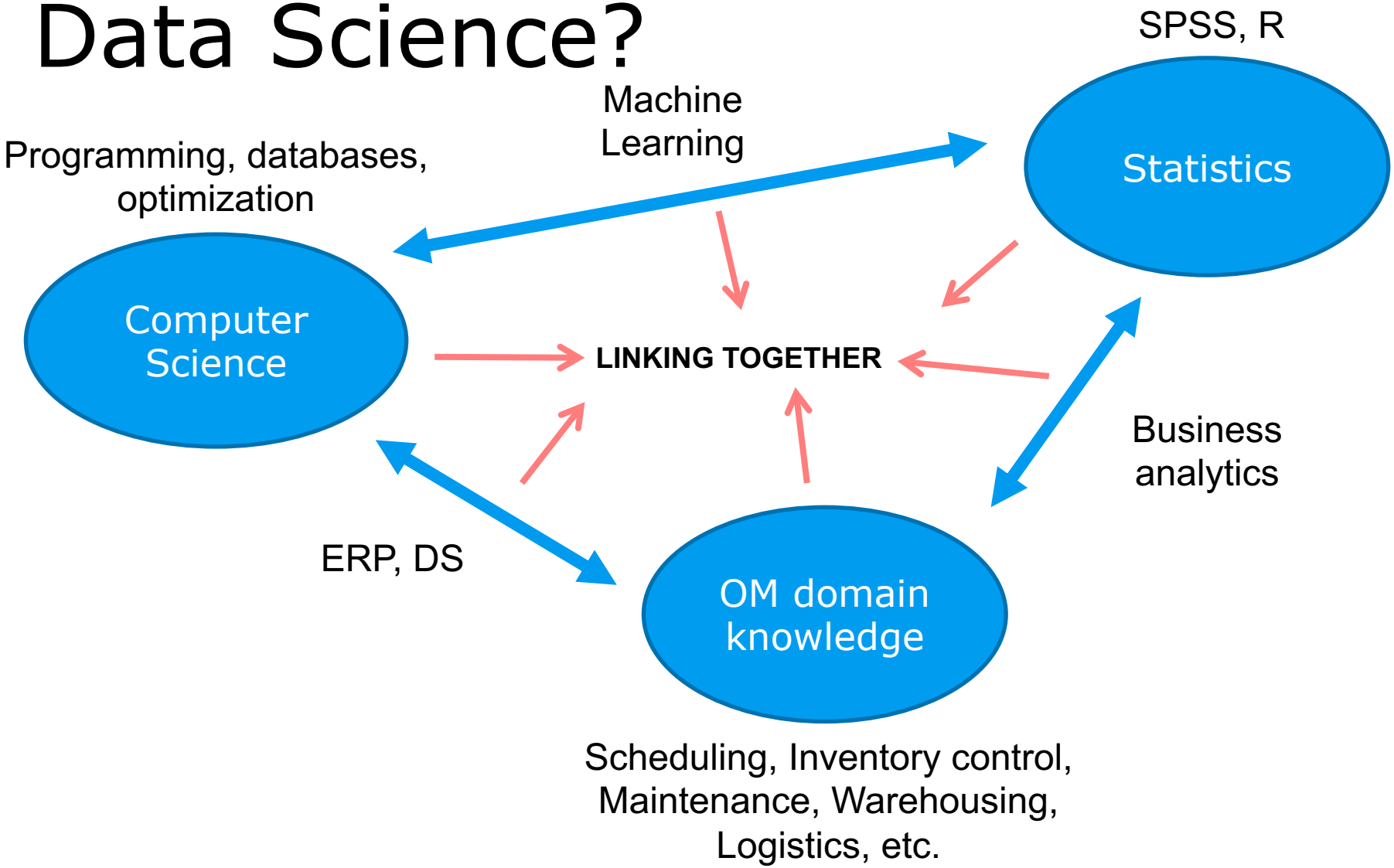
# Contents

- Why, What, How, and When of Dapom

# Why Dapom?

- Recent developments: organizations start generating Big Data
  - Marketing, user profiling, etc.
  - Supply Chain Integration
  - Horizontal collaboration in transportation
  - Internet of Things: individual machines that collect data
- Generic Data Scientists (BSc/MSc in computer science / artificial intelligence) don't understand Operations Management
- OM theory, concepts, and tools to include Big Data are not yet standardized

# Data Science?



# Learning goals

- You will learn how you can:
  - Create a computer program
  - Solve optimization models
  - Handle big data
- You can use these skills in other courses, in your thesis, and in your future jobs

# Learning goals

- We are not a computer science program.
- Main design goals regarding programming skills:
  - Very important skillset if you need to do simple tasks with data (for example, combining data sets that don't fit in Excel)
  - You don't need to be experts, but you should be able to recognize experts.
  - Maybe you won't program yourself in your future job, but you must be able determine what is technically possible and what a reasonable price tag is if you hire a programmer.

# Learning goals

- Programming is more a mindset than a skill.
- Essentially, programming is really easy.

- We have:
  - Variables

    ```
    a=3, b=2, c=a+b
    ```
  - Branching based on conditions

    ```
    if a > 4 then a=a+2 else a=a+3
    ```
  - Iteration

    ```
    while a<b do a=a*2
    ```
  - Encapsulation (procedures, objects, libraries): combine multiple programming commands to logical units that you can reuse.

# Learning programming

- The challenge is to get your mind working a bit like a computer. You must learn how to visualize variables, branching, and iteration.

- You learn this by iteratively:
  - Doing (including failing)
  - Looking at how others solved their problems

- You don’t learn if you copy code and confirm that it works. You need to understand why it works, otherwise you’ll get in to problems when things get more complex.

# Role in TOM program

- Prerequisite knowledge:
  - BSc Operations Management
  - Statistics
- DAPOM
  - Programming, optimization, big data
- You can use DAPOM skills in other TOM courses and in your thesis
  - warehousing, scheduling, simulation, inventory management, maintenance, energy transition

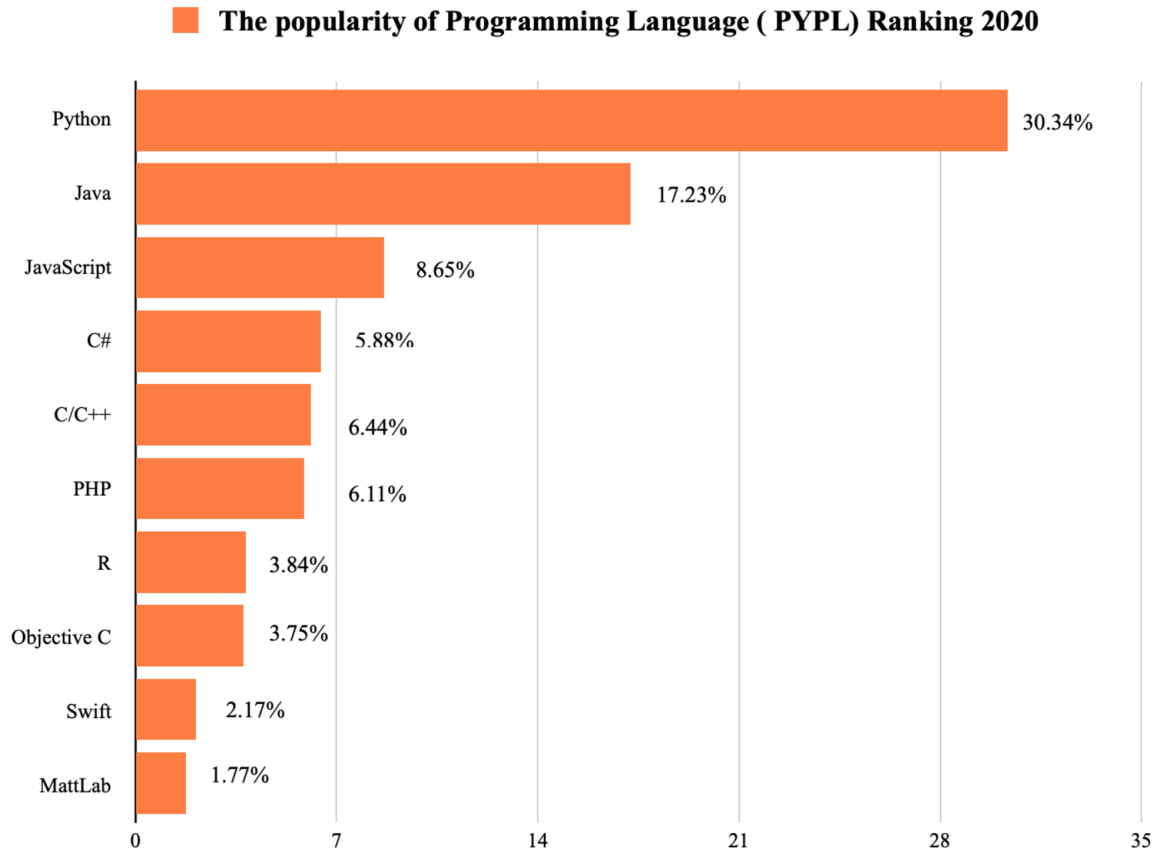
# OMC

- Operations Management & Control course
- Uses programs to show the effects of parameters
- Programs are provided by the lectureres

# Examples in theses

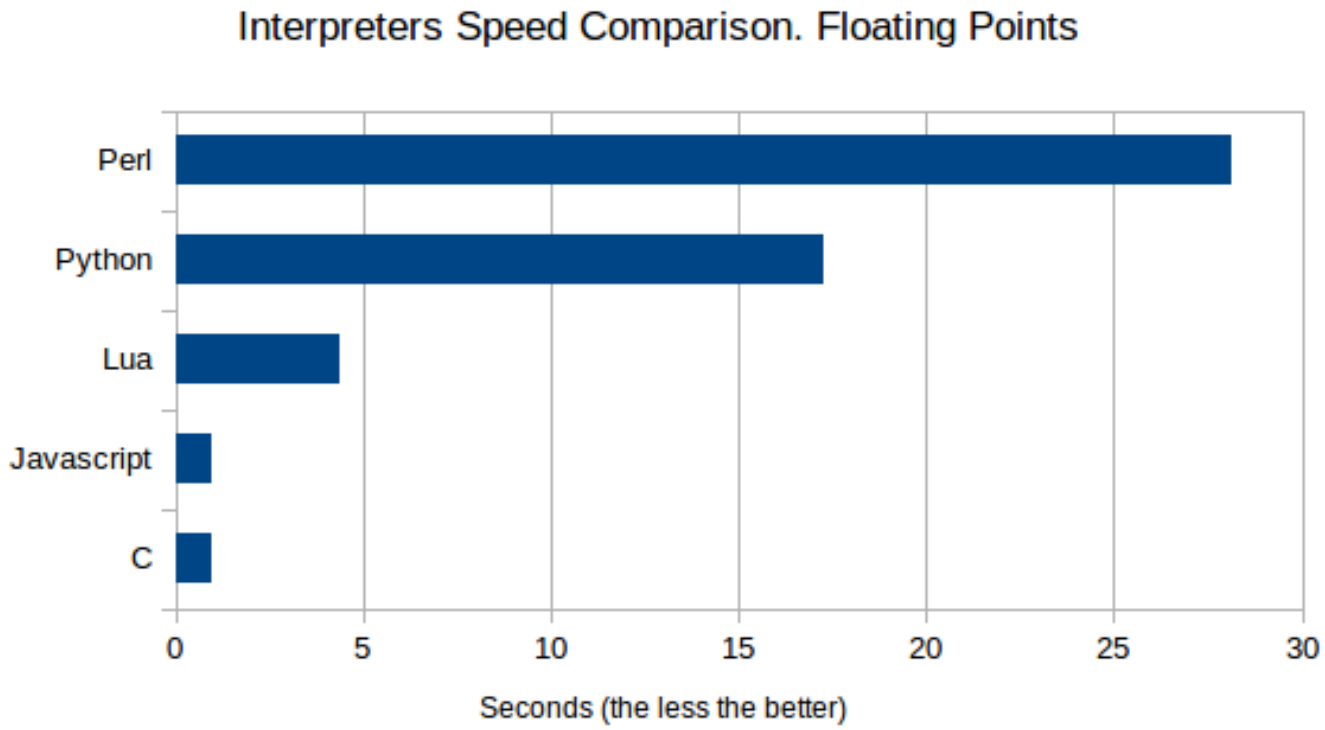
- Bram de Brabander: Production scheduling system for Philips
  - Usable factors of scheduling systems
- Max Cornel: Data analysis of 7 years of shift data of 2200 engineers at KLM, looking at health of schedules
  - Staff schedules should focus on when you rest, not when you work
- Jamil Karchoud: Neural network to detect production errors at Luxaflex (window blinds)
  - Using image recognition in a factory
- Esmee vd Berg: Blockchain for data security in logistics (Physical Internet)

# How?




https://darly.solutions/the-most-popular-programming-languages-in-2021/

# Why Python?


Interpreters Speed Comparison. Floating Points
Perl
Python
Lua
Javascript
C
0
5
10
15
20
25
30
Seconds (the less the better)

# Why Python?

- Easy to learn
- Very good 'ecosystem': development tools, libraries, communities
- Lots of people use it, so many code examples and answers to questions can be found online
- Performance intensive tasks are done by libraries (for example, optimization in Gurobi). These libraries are typically written in the programming language C and very fast

# Be language-agnostic

All programming languages support variables, branching, iteration, encapsulation. Only the syntax is different.

```
if sunHours > 3 then
begin
  goodWeather:=goodWeather+1;
end
else
begin
  badWeather:=badWeather+1;
end;
```

```
if (sunHours > 3) {
 goodWeather=goodWeather+1;
}
else
{
  badWeather=badWeather+1;
}
```

```
if sunHours > 3:
  goodWeather=goodWeather+1
else:
  badWeather=badWeather+1
```

# How?

- We will use the programming language Python and appropriate libraries to go through the various data science steps:
  - Data collection (e.g., machine failure data)
  - Data manipulation (e.g., filtering, aggregation)
  - Data exploration & analysis (statistics, machine learning)
  - Optimization (routing, maintenance intervals, scheduling). This transcends traditional data science!

# When?

- Weekly chapters in Manuals
- Q&A sessions
- Weekly on-site practicals with assignments to work together and ask questions. The practicals are there mainly to teach you the mindset for programming.
- Assessment: big individual final assignment, and an oral defense.

# Your weekly process

- In the days before the practical
  - Read manual A (generic programming)
  - Do assignments from manual A
  - Read manual B (OM application)
  - Do assignments from manual B

- Continue with assignments at the practical
- Discuss with teacher and fellow students where you got stuck
- Continue working on the assignments after the practical

- Send questions about course literature or generic programming questions before the Q&A

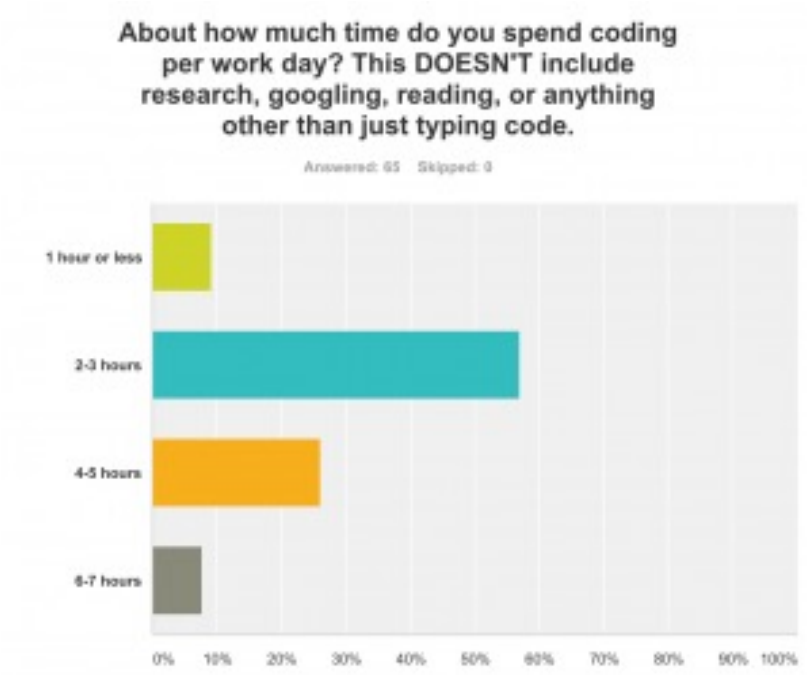
# Warning

- If you start with the weekly material at the practical, you will probably not yet get into problems during the practical.
- This means you would not be able to make efficient use of our help.
- Note: especially later in the course, we will not always be able to spot the cause of a problem immediately.

# Last note

# Last note

- Programming is (approximately):
  - 35% thinking how to attack a problem, of which 80% is spent on searching how others attacked it.
  - 20% to search for existing code.
  - 15% to understand the existing code.
  - 30% to code yourself, of which 40% is spent on debugging, and 40% on performance profiling and improving.

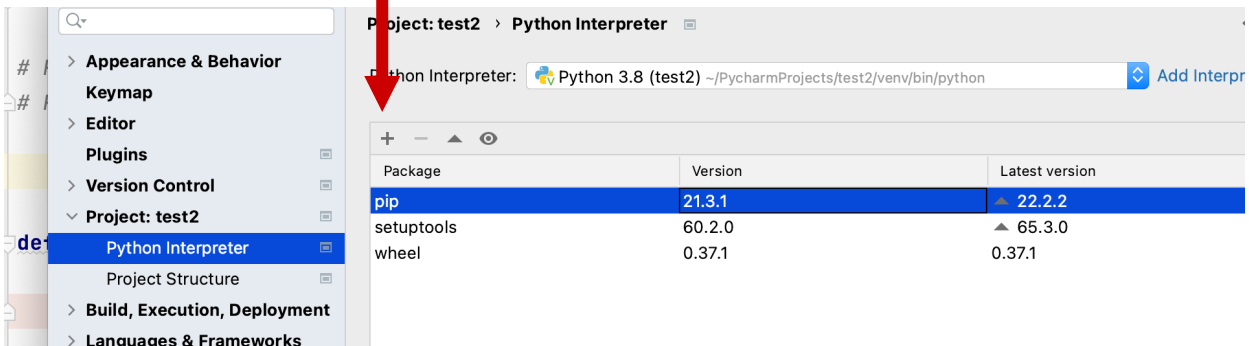



Also see: https://sweetcode.io/developers-really-spend-time-hint-not-coding/

# Part 2: Installfest

- Download and install PyCharm community edition.
- On the Mac, make sure to choose the Intel version.
- On the Mac, when you create a default project, it might propose to you that the Command-Line Developer Tools are installed. Click Ok for this.
- Try to run the basic 'Hi, Pycharm' script. If it does not work, check if you need to install Python yourself (https://www.python.org/downloads/)

- Install the following packages: numpy, pandas, matplotlib, scipy, xlrd, openpyxl
  - In Pycharm: go to File-Settings (windows) or Preferences (Mac) choose your Project, and then Python Interpreter.
    - Click on the + to install packages; you get a screen in which you can search for them.



# Part 2: Installfest

- Install Gurobi, see practical manual B.
- On the Mac, make sure to use the Intel version (gurobi9.1.2_mac64.pkg). You can find it on the download page.

- You can link your Python environment to Gurobi using Pycharm. For this, install the Gurobipy package (see previous slide how to install packages).
- If this works, then run the product_mix.py program you can find on Brightspace.
- If you get correct output (Obj. 5575.94), then you can skip the 'python setup.py install' step which is mentioned in the practical manual. Otherwise, you need to perform that step manually.

› We wish you happy coding!