Fundamentos de Programação 2025-2026

Programming Fundamentals

What do you hope to learn during this course?

What topics are you most curious to tackle in this course?

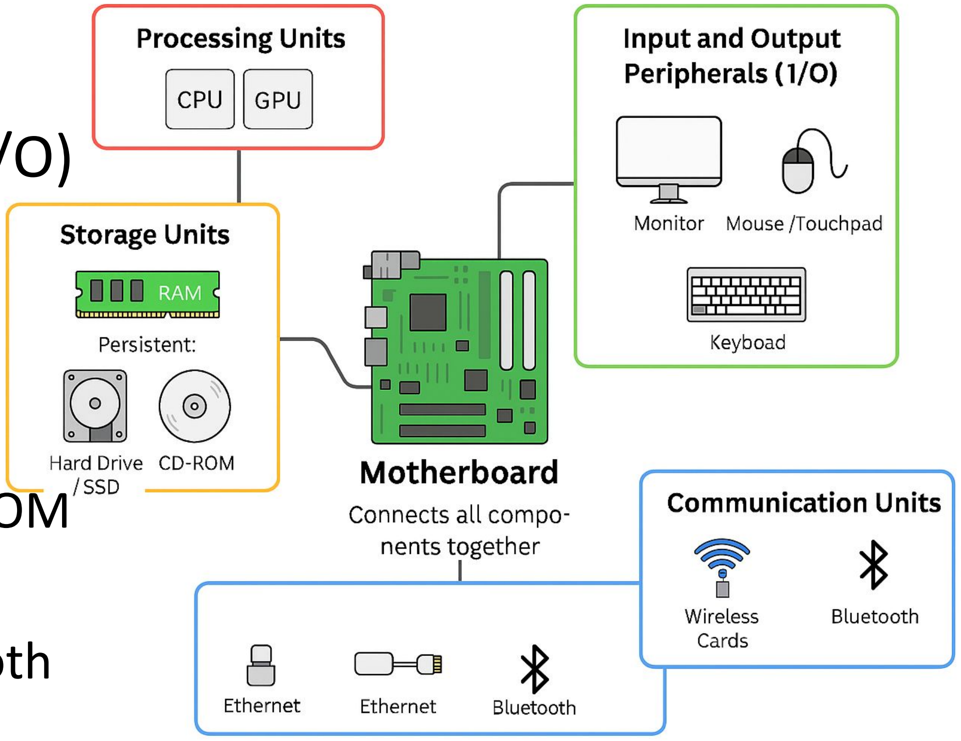
Overview

- Computers
- What is programming
- Programming = Problem Solving
- Programming in AI Era
- Python
- Values and Types
- Variables
- Input/output

COMPUTERS BASICS

The computer

- Processing Units
  - CPU, GPU
- Input and Output Peripherals (I/O)
  - Monitor, Mouse/Touchpad, Keyboard
- Storage Units
  - Volatile: RAM
  - Persistent: Hard Drive/SSD, CD-ROM
- Communication Units
  - Wireless cards, Ethernet, Bluetooth
- Motherboard
  - Connects all components together.



Processing Unit (CPU & GPU)

- CPU: Central Processing Unit
  - Usually has multiple processing cores (1, 2, 4, ...)
  - Several levels of internal memory (cache): L1, L2, L3
- Essentially: they **transfer** and **operate** on data:
  - Store and retrieve data from memory
  - Add, subtract, multiply, divide
  - Compare values (< , =, >)
- Execute instructions **sequentially**.
- But they **can jump** to instructions behind or ahead.
- They can also execute **conditionally**:
  - If x < 0, do this; otherwise, do that

Memory

- The smallest unit of memory can only distinguish between two states
  - *Charged or discharged, on/off, 1 or 0.*
- It is called a **bit** *(short for binary digit).*
- A group of 8 bits is called a **byte**.

- Computer memory contains many bytes, and each one has a unique **address**
  - *usually expressed as a hexadecimal number.*
- All data (numbers, text, images, etc.) is encoded and stored in groups of bits.
- The encoding depends on the characteristics (**type**) of data.

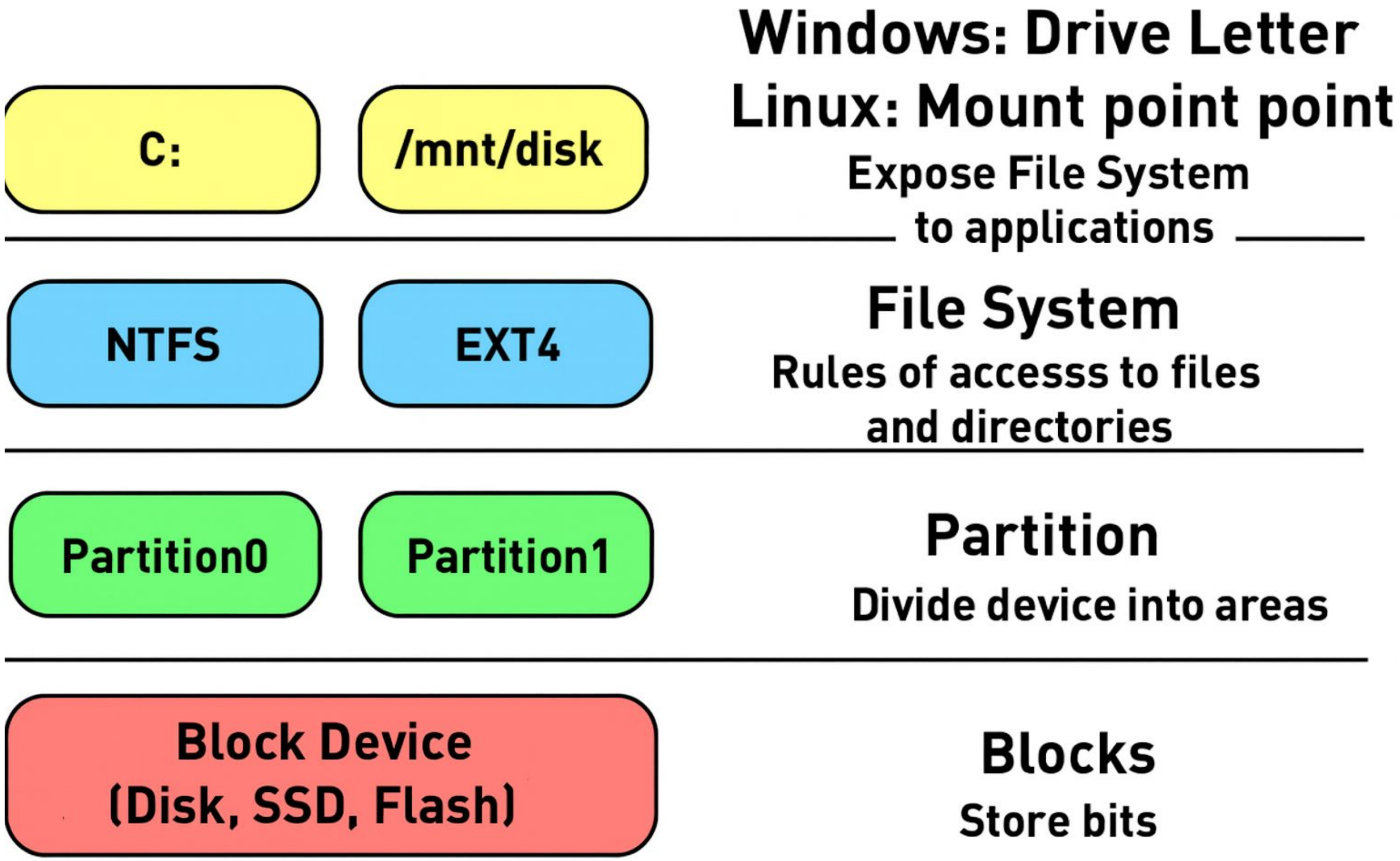
Data representation in Memory

- Example:
  - The integer number $853_{10} = 11\ 0101\ 0101_2$
    - (stored in 16 bits, little-endian)
  - The text “UV853” (5 characters, 1 byte each)

Endereço	Byte	Dados	Tipo
FC000000	0101 0101	853	int, 16 bit, little-endian
FC000001	0000 0011		
FC000002	0101 0101	'U'	char
FC000003	0101 0110	'V'	char
FC000004	0011 1000	'8'	char
FC000005	0011 0101	'5'	char
FC000006	0011 0011	'3'	char

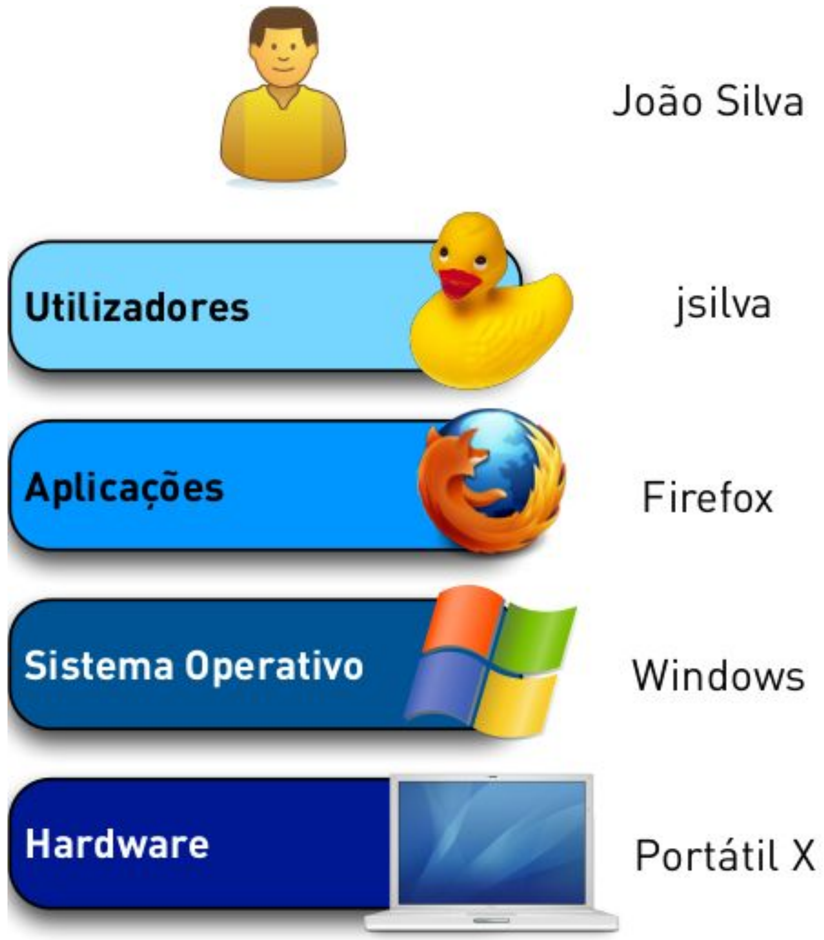
- **The number 853 and the text “853” are represented with very different bytes!**

File organization



Operating System

- Program executed by a processor
  - With direct access to hardware
- OS Manages:
  - Hardware
  - Applications
  - File System
  - Memory



PROGRAMMING

Why Programming?

Applications: automation, data science, AI, games, …

Professional career, research, problem solving …

Why Programming in Engineering?

- Problem-Solving Tool:
  - Engineers face complex problems.
  - Programming provides a systematic way to model, simulate, and solve them.
- Automation & Efficiency:
  - Replaces repetitive manual tasks with automated solutions.
  - Saves time, effort, and resources.
- Data Analysis & Decision Making:
  - Engineers work with large amounts of data.
  - Programming allows for processing, visualization, and extracting insights.
- Design & Simulation:
  - Used in CAD, circuit design, simulations (fluids, mechanics, electronics).
  - Test solutions virtually before building real prototypes.
- Interdisciplinary Power:
  - Programming links engineering with AI, IoT, robotics, and big data.

- A must-have skill for modern engineers.

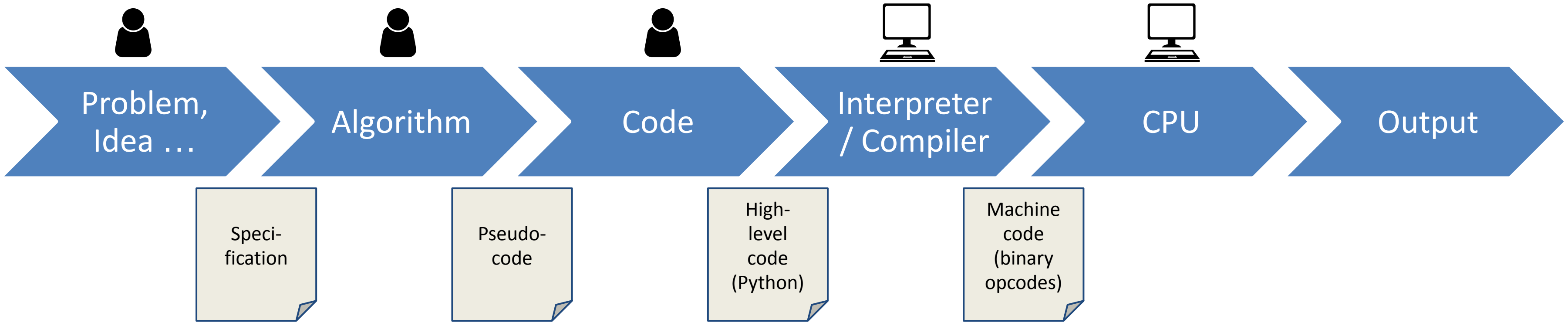
Programming

Programming is not just about syntax

Programming is about problem solving, and critical thinking

Programming is not creating code (without a purpose)

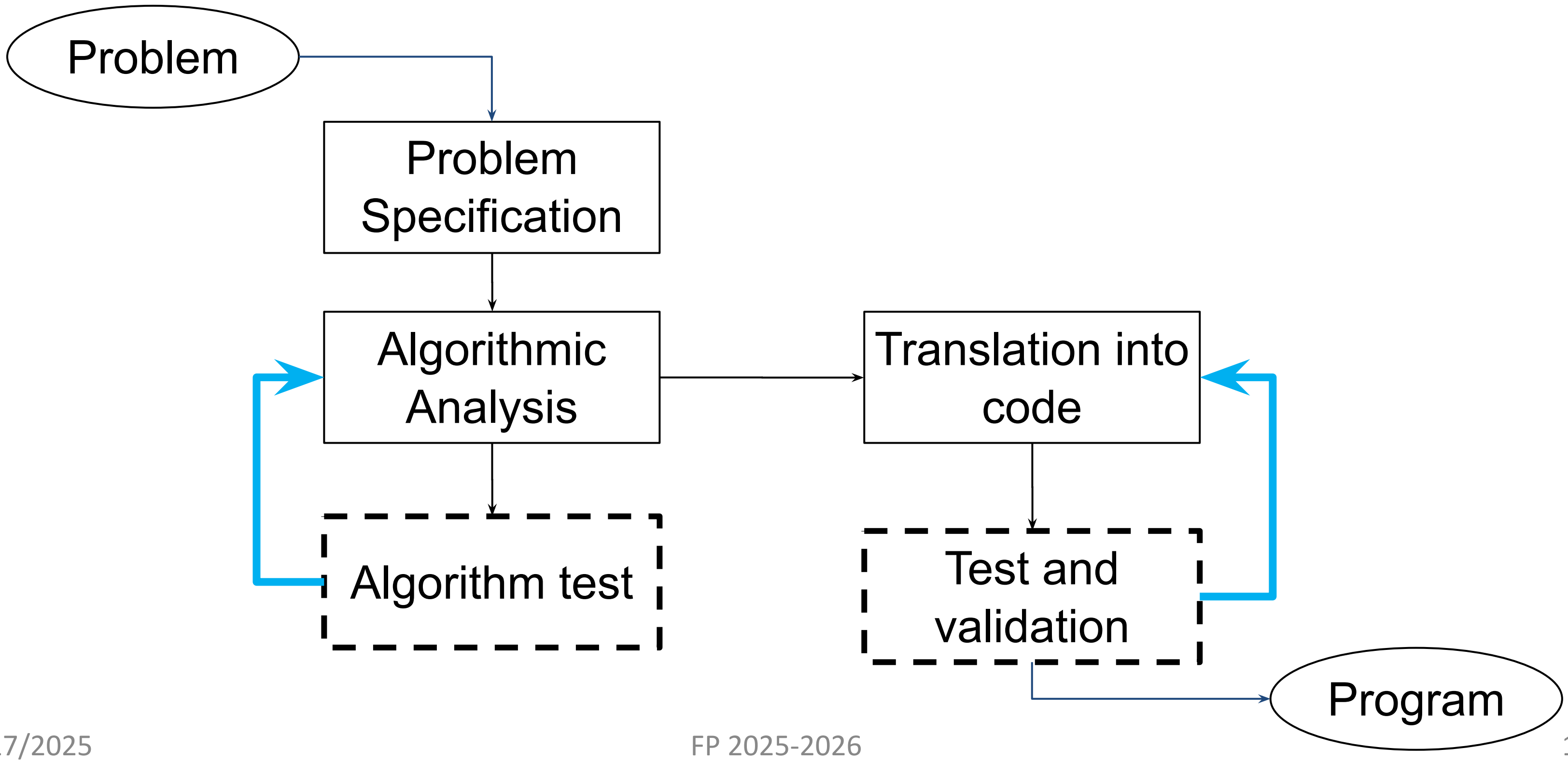
Programming



- Humans understand high-level programming languages such as Python, Java, or C.
- Computers understand machine code, a binary encoding of CPU operations (opcodes).
- A program (interpreter or compiler) translates high-level code into machine code.

Phases of program development

- The two essential stages of program development are **problem analysis** and **application implementation**.



SW Development Process

THE MOST SIMPLIFIED OVERVIEW OF THE
SDLC
SOFTWARE DEVELOPMENT LIFECYCLE

https://www.youtube.com/watch?v=mmVXL0LzLks

Important

- Before producing code, we need to **understand the problem** and think about **how to solve it**.

- We need an **algorithm**.

Errors and debugging

- Programming errors are called **bugs**.
- Tracking down and correcting bugs is called **debugging**.
- There are three kinds of errors:

- Syntax errors: occur if the program contains code that does not respect the syntactical rules of the programming language

- Runtime errors: only appear after the program has started running.
  - They are often caused by type mismatches or failure to deal with special cases (such as division by zero).
  - These errors are also called exceptions.

- Semantic errors: when a program runs with no error messages, but still produces wrong results.
  - The program is not doing what the programmer intended.
  - It is doing exactly what it was told to do.

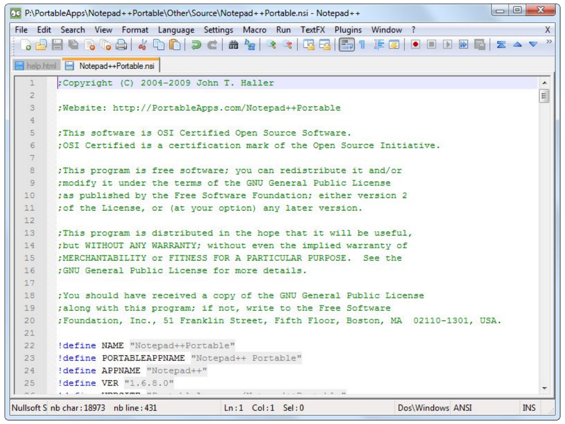
Documenting the code is essential

- In real word, programming is a team effort
- Programs need to be updated, improved, maintained …
- It is **highly recommended to add notes** to a program to explain in natural language what the program is doing.

Tools

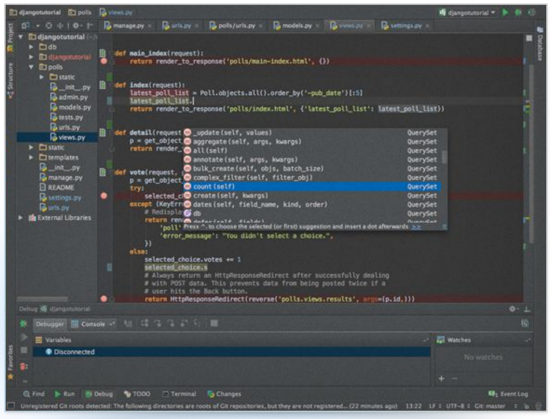
geany
gedit
VSCode
Notepad++
PyCharm

VIM
Nano



```
;Copyright (C) 2004-2009 John T. Haller

;Website: http://PortableApps.com/Notepad++Portable

;This software is OSI Certified Open Source Software.
;OSI Certified is a certification mark of the Open Source Initiative.

;This program is free software; you can redistribute it and/or
;modify it under the terms of the GNU General Public License
;as published by the Free Software Foundation; either version 2
;of the License, or (at your option) any later version.

;This program is distributed in the hope that it will be useful,
;but WITHOUT ANY WARRANTY; without even the implied warranty of
;MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
;GNU General Public License for more details.

;You should have received a copy of the GNU General Public License
;along with this program; if not, write to the Free Software
;Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301, USA.

!define NAME "Notepad++Portable"
!define PORTABLEAPPNAME "Notepad++ Portable"
!define APPNAME "Notepad++"
!define VER "1.6.8.0"
```

P:\PortableApps\Notepad++Portable\Other\Source\Notepad++Portable.nsi - Notepad++
File Edit Search View Format Language Settings Macro Run TextFX Plugins Window ?
help.html
Notepad++Portable.nsi
Nullsoft S nb char : 18973 nb line : 431 Ln : 1 Col : 1 Sel : 0 Dos\Windows ANSI INS



```
def main_index(request):
    return render_to_response('polls/main-index.html', {})


def index(request):
    latest_poll_list = Poll.objects.all().order_by('-pub_date')[:5]
    latest_poll_list.
    return render_to_response('polls/index.html', {'latest_poll_list': latest_poll_list})


def detail(request
    p = get_object
    return render_


def vote(request,
    p = get_object
    try:
        selected_c
    except (KeyErr
        # Redispla
        return ren
            'poll'
            'error_message': "You didn't select a choice.",
        })
    else:
        selected_choice.votes += 1
        selected_choice.s
        # Always return an HttpResponseRedirect after successfully dealing
        # with POST data. This prevents data from being posted twice if a
        # user hits the Back button.
        return HttpResponseRedirect(reverse('polls.views.results', args=(p.id,)))
```

```
_update(self, values)                    QuerySet
aggregate(self, args, kwargs)            QuerySet
all(self)                                QuerySet
annotate(self, args, kwargs)             QuerySet
bulk_create(self, objs, batch_size)      QuerySet
complex_filter(self, filter_obj)         QuerySet
count(self)                              QuerySet
create(self, kwargs)                     QuerySet
dates(self, field_name, kind, order)     QuerySet
db                                       QuerySet
Press ^. to choose the selected (or first) suggestion and insert a dot afterwards >>
```

VERY BRIEF HISTORIC NOTES

Quiz

- Who is known for developing the concept of an algorithm and is often called the “father of theoretical computer science”?
  - a) John von Neumann
  - b) Alan Turing
  - c) Charles Babbage
  - d) Dennis Ritchie
- Which early programmer is often credited as the first computer programmer for writing an algorithm for the Analytical Engine?
  - a) Grace Hopper
  - b) Ada Lovelace
  - c) John Backus
  - d) Margaret Hamilton
- What was the name of the first general-purpose electronic computer built in the 1940s?
  - a) UNIVAC
  - b) Z3
  - c) ENIAC
  - d) Colossus
- Which programming language, developed in the 1950s, is one of the earliest high-level languages and is still in use today, especially in scientific computing
  - a) COBOL
  - b) Pascal
  - c) Fortran
  - d) BASIC

Quiz

- Who is known for developing the concept of an algorithm and is often called the “father of theoretical computer science”?
  a) John von Neumann
  b) Alan Turing
  c) Charles Babbage
  d) Dennis Ritchie
- Which early programmer is often credited as the first computer programmer for writing an algorithm for the Analytical Engine?
  a) Grace Hopper
  b) Ada Lovelace
  c) John Backus
  d) Margaret Hamilton
- What was the name of the first general-purpose electronic computer built in the 1940s?
  a) UNIVAC
  b) Z3
  c) ENIAC
  d) Colossus
- Which programming language, developed in the 1950s, is one of the earliest high-level languages and is still in use today, especially in scientific computing
  a) COBOL
  b) Pascal
  c) Fortran
  d) BASIC

Quiz (cont.)

- What does the acronym “COBOL,” an early programming language widely used in business, stand for?
  a) Common Business Operational Logic
  b) Common Business-Oriented Language
  c) Computer Binary Oriented Language
  d) Centralized Business-Oriented Logic

- Who created the Python programming language in the early 1990s?
  a) James Gosling
  b) Bjarne Stroustrup
  c) Guido van Rossum
  d) Dennis Ritchie

- What is the name of the operating system created at AT\&T’s Bell Labs in the 1970s that influenced many modern systems like Linux and macOS?
  a) Windows
  b) CP/M
  c) Unix
  d) DOS

- Which programming language, created in 1995, is famous for its “write once, run anywhere” philosophy?
  a) Python
  b) C++
  c) Java
  d) Ruby

Quiz (cont.)

- What does the acronym “COBOL,” an early programming language widely used in business, stand for?
  a) Common Business Operational Logic
  [✓ b) Common Business-Oriented Language]
  c) Computer Binary Oriented Language
  d) Centralized Business-Oriented Logic

- Who created the Python programming language in the early 1990s?
  a) James Gosling
  b) Bjarne Stroustrup
  [✓ c) Guido van Rossum]
  d) Dennis Ritchie

- What is the name of the operating system created at AT\&T’s Bell Labs in the 1970s that influenced many modern systems like Linux and macOS?
  a) Windows
  b) CP/M
  [✓ c) Unix]
  d) DOS

- Which programming language, created in 1995, is famous for its “write once, run anywhere” philosophy?
  a) Python
  b) C++
  [✓ c) Java]
  d) Ruby

Why This Matters?

- You’re Part of the Story Now

- You are the future creators
  - Potential new Larry Page (Google), Bill Gates (Microsoft), …

- This course aims to teach you **how to think** (like a programmer)

Origins of Programming - From Mechanical Dreams to Digital Reality

- 1800s: Ada Lovelace writes the first algorithm
- 1930s–40s: Alan Turing defines the theoretical model of computation
- 1940s: ENIAC: first electronic computer
  - programmed with switches and punch cards

- Early computing was more about theory and hardware than software

The Rise of Programming Languages - Making Machines Understand Us

- 1950s: FORTRAN (scientific), COBOL (business)
- 1960s–70s: Structured programming
  - ALGOL, Pascal, C

- The shift from machine code to human-readable languages

Paradigm Shifts - New Ways to Think About Code

- 1980s: Object-Oriented Programming:
  - Smalltalk, C++
- 1990s: Internet boom
  - Java, JavaScript, PHP
- 2000s: Scripting languages
  - Python, Ruby

- Each paradigm changed how we approach problems

Programming in the AI Era - A new age of computing

- 2010s–Today: AI-assisted coding:
  - GitHub Copilot, ChatGPT
  - Programming becomes collaborative and creative
  - Focus shifts from writing code to designing intelligent systems

- AI is transforming the role of the programmer

Why Learn Programming Today?

- Programming is becoming a **universal skill**
  - like reading or math
- It empowers you to **build, automate, and create** in any field.
  - from art to science

- AI should be your assistant, you the architect.
- To be the architect you need to learn how-to solve problems.

RECOMMENDED AI USAGE

Responsible use of AI in this course

- Use AI as a teacher not as a replacement for your duties
  - ChatGPT and Copilot and similar tools can be used to explain, suggest, create exercises
- Don't copy blindly — understand the code
- Results not always correct ⟶ Always verify and assess AI results

- Use AI as a partner, not a substitute
  - Interpretation of AI as Augmented Intelligence
    - Your intelligence

What is Generative AI?

- What is Gen AI?
- What are its **limitations**?
- How to use it **responsibly**?
- This and more on [Intro to GenAI (TU Dublin)](#).

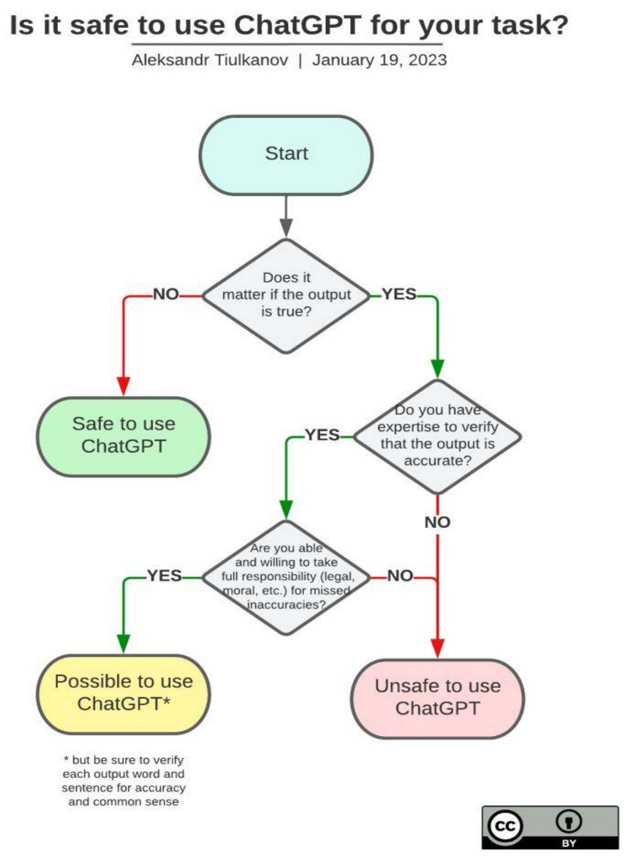

Is it safe to use ChatGPT for your task?
Aleksandr Tiulkanov | January 19, 2023
Start
Does it matter if the output is true?
NO
YES
Safe to use ChatGPT
Do you have expertise to verify that the output is accurate?
YES
NO
Are you able and willing to take full responsibility (legal, moral, etc.) for missed inaccuracies?
YES
NO
Possible to use ChatGPT*
Unsafe to use ChatGPT
* but be sure to verify each output word and sentence for accuracy and common sense

Illustration by: [Aleksandr Tiulkanov LL.M. @ Linkedin](#).

A new age of computing programming and learning how to program

- AI doesn't replace programmers, it augments them
- Understanding how solutions and code work is much more important than memorizing it

- 🤖 Humans + Machines = Superpowers

Using AI as a Programming Assistant – Prompt examples

1. Explain [New Python Topic] in simple terms, like you're talking to someone who is just starting to learn programming. Focus on what it is, why we use it, and its main purpose.

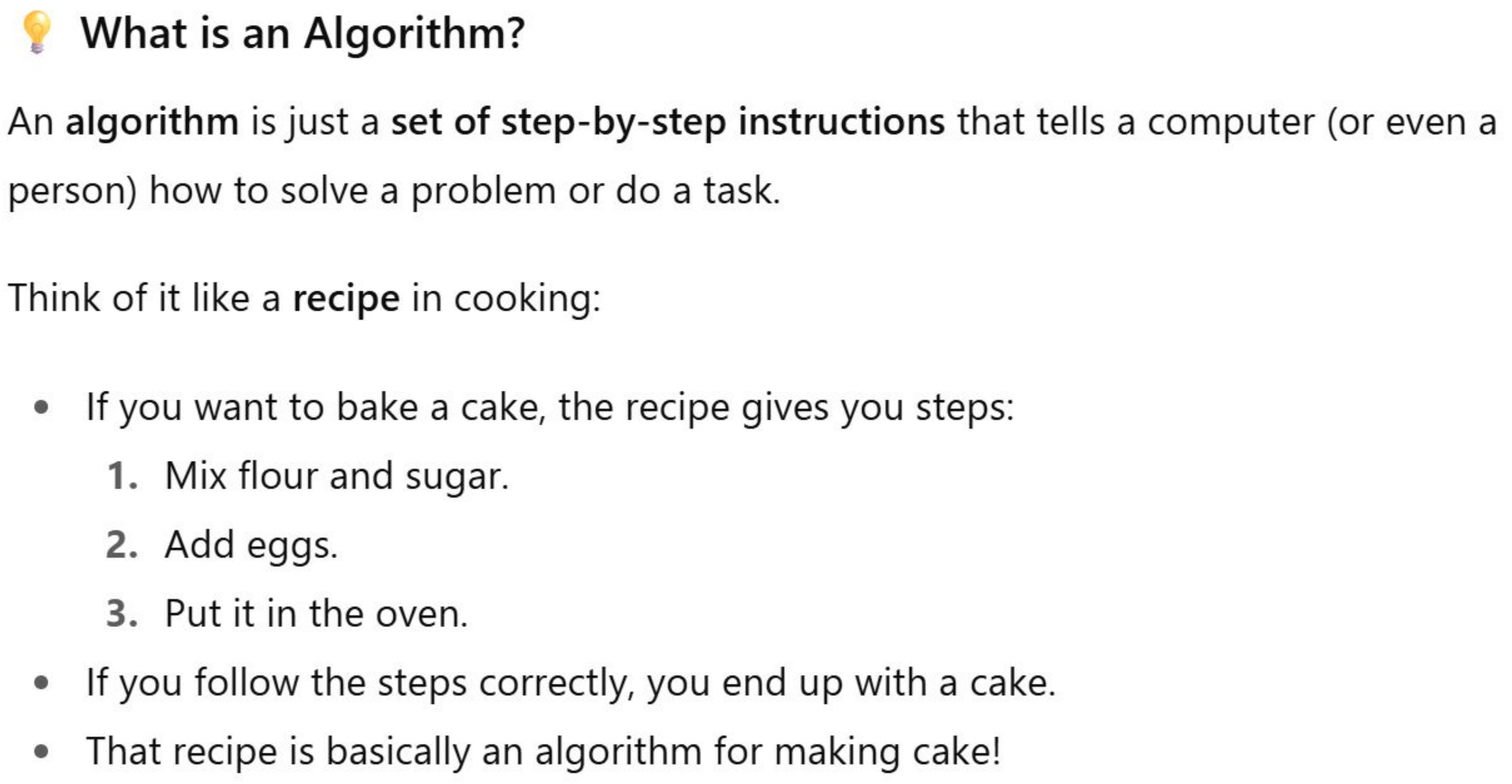
2. Show me a basic working example of [New Python Topic]. Explain each line or significant part of the code step-by-step so I can understand exactly what's happening.

Using AI as a Programming Assistant – Prompt examples

4. Give me a small coding exercise related to [New Python Topic]. Describe the problem I need to solve using this concept, and maybe offer a hint if I get stuck.

5. What are some common mistakes beginners make when using functions? How can I identify and fix them?

6. Suggest a slightly more complex scenario or a mini-project idea where I could effectively use [New Python Topic] along with other basic concepts I might already know.



💡 What is an Algorithm?

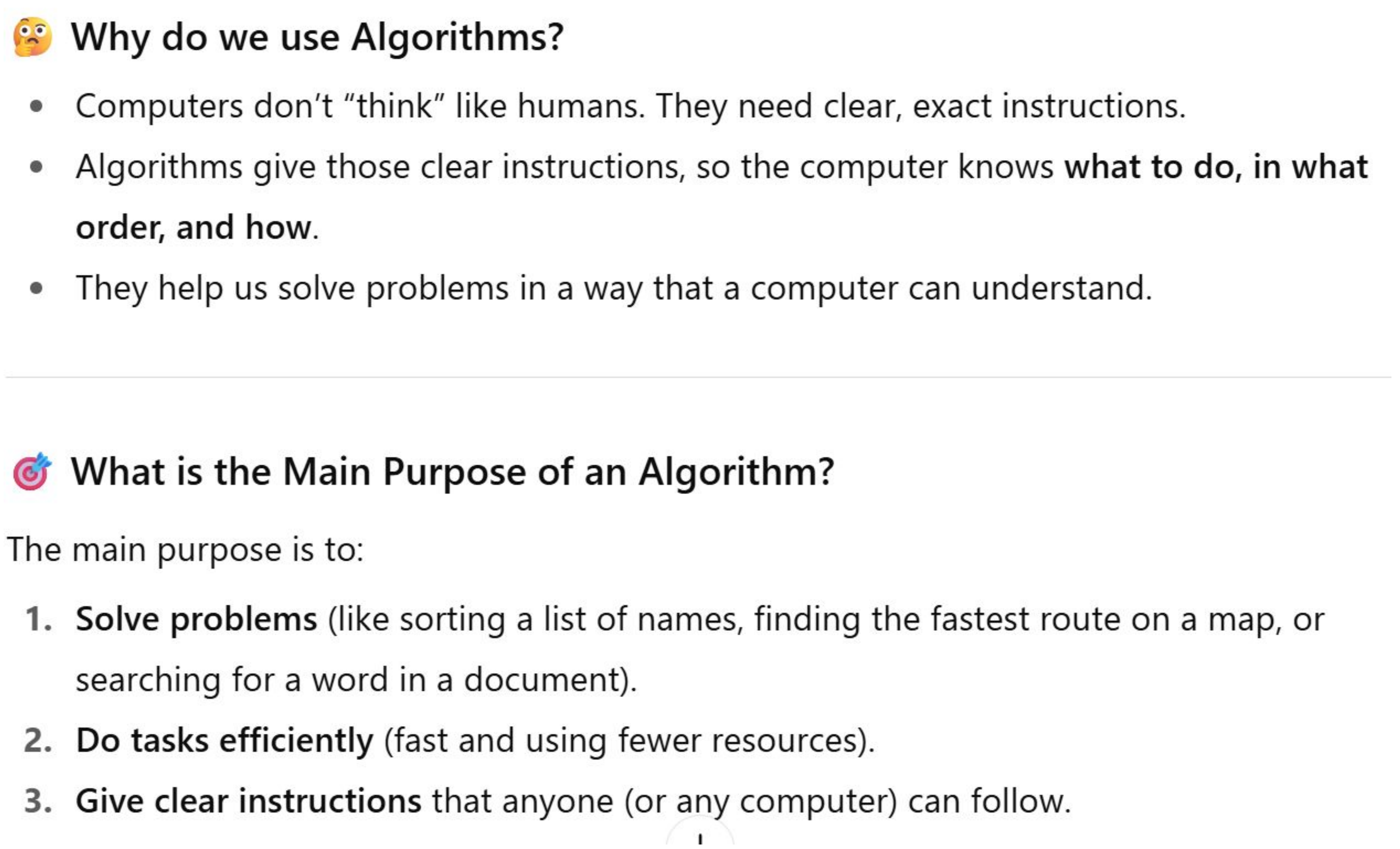
An **algorithm** is just a **set of step-by-step instructions** that tells a computer (or even a person) how to solve a problem or do a task.

Think of it like a **recipe** in cooking:

- If you want to bake a cake, the recipe gives you steps:
  1. Mix flour and sugar.
  2. Add eggs.
  3. Put it in the oven.
- If you follow the steps correctly, you end up with a cake.
- That recipe is basically an algorithm for making cake!

Results ChatGPT

[Prompt 1] Explain algorithms in simple terms…



🤔 Why do we use Algorithms?

- Computers don't "think" like humans. They need clear, exact instructions.
- Algorithms give those clear instructions, so the computer knows **what to do, in what order, and how**.
- They help us solve problems in a way that a computer can understand.

🎯 What is the Main Purpose of an Algorithm?

The main purpose is to:

1. **Solve problems** (like sorting a list of names, finding the fastest route on a map, or searching for a word in a document).
2. **Do tasks efficiently** (fast and using fewer resources).
3. **Give clear instructions** that anyone (or any computer) can follow.

Results ChatGPT

[Prompt 1] Explain algorithms in simple terms…

APPROACHES TO PROGRAMMING

Programming Paradigms

- There are several different ways to solve problems (with code)

- Paradigm = **style of thinking about problems and structuring programs**

- A language can support multiple paradigms (Python does)

Imperative Programming

- Tell the computer how to do things, step by step
- C, Python, Java

- Example:

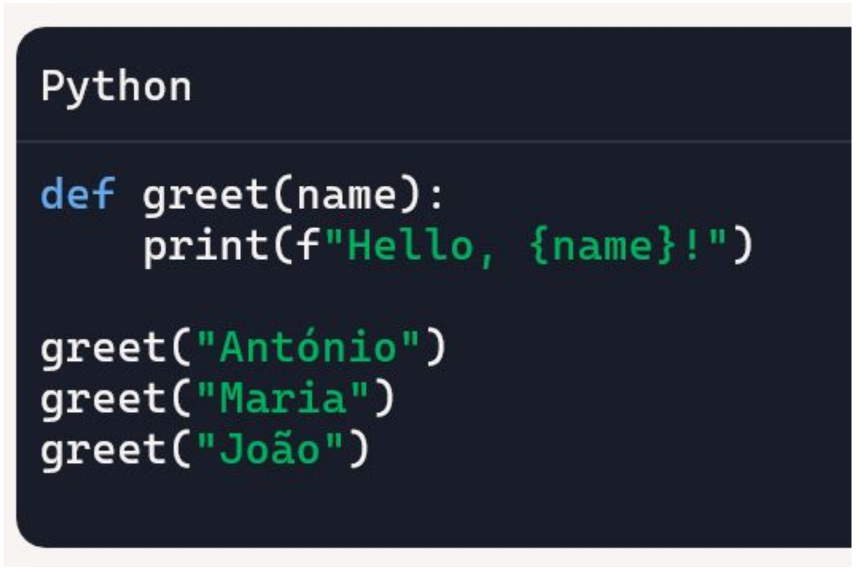
```
for i in range(5):
print(i)
```

- Variables, loops, conditionals
- Suitable for:

Procedural Programming

- Subtype of imperative programming
- Organize code into procedures (functions)
  - Subprograms
- Promotes reuse and modularity
- Example:

```
def square(x):
        return x*x
```



Python

```python
def greet(name):
    print(f"Hello, {name}!")

greet("António")
greet("Maria")
greet("João")
```

- How and why functions were created? When? By whom?

Declarative Programming

- Tell the computer what result you want
  - not how
- Focus on describing rules and relations
- SQL, HTML, Prolog

- Example:

```
SQL SELECT name FROM Students WHERE grade > 15;
```

Object-Oriented Programming (OOP)

- Model real-world entities as objects (data + behavior)
- Python example:

```
class Dog:
    def bark(self):
        print('Woof!')
```

- Common in large-scale applications
- Principles: encapsulation, inheritance …

Functional Programming

- Emphasizes functions as first-class citizens
- Haskell, Lisp, also supported in Python

- Python example:

```
squared = list(map(lambda x: x*x, nums))
```

Logic Programming

- Based on formal logic
- Define rules and let the engine infer solutions

- Example: Prolog

- Python does not directly support, but useful conceptually

Programming Paradigms – Pros, Cons & Suitability

Paradigm	Pros	Cons	Best Suited For
Imperative	Simple and intuitive (step-by-step) Direct control of program state	Can become messy with large codebases Harder to reason about state changes	Small to medium programs, scripting, system programming
Declarative	Focus on what not how Often more concise Closer to problem domain	Less control over execution Sometimes less efficient Harder to debug	Databases (SQL), configuration (HTML, CSS), rule-based systems
Procedural	Organizes code into reusable functions Easier to test and maintain Clear program flow	Relies on mutable state Can get complex with very large systems	General-purpose apps, scientific computing, structured programming
Object-Oriented (OOP)	Models real-world entities Encapsulation, inheritance, polymorphism Promotes reuse and modularity	Can be over-engineered May introduce complexity for small tasks	Large software systems, GUIs, simulations, enterprise apps
Functional	Emphasizes immutability → fewer bugs Easier reasoning with pure functions Powerful abstractions (map, reduce)	Less intuitive for beginners Can be verbose Performance overhead sometimes	Data processing, parallel/distributed computing, AI/ML pipelines
Logic	Expressive for knowledge representation Great for search, inference, constraint solving	Less efficient Steeper learning curve Limited real-world adoption	AI (expert systems), constraint satisfaction problems, research

Paradigms in Python

- Python is multi-paradigm:
  - Imperative → loops, conditionals
  - Procedural → functions
  - OOP → classes, objects
  - Functional → lambdas, map/filter/reduce

- Flexibility depending on the problem

Why Paradigms Matter

- Different paradigms suit different problem types
- Understanding paradigms helps choose the right approach
- Improves problem solving and programming flexibility

Defining solutions for problems

ALGORITHMS

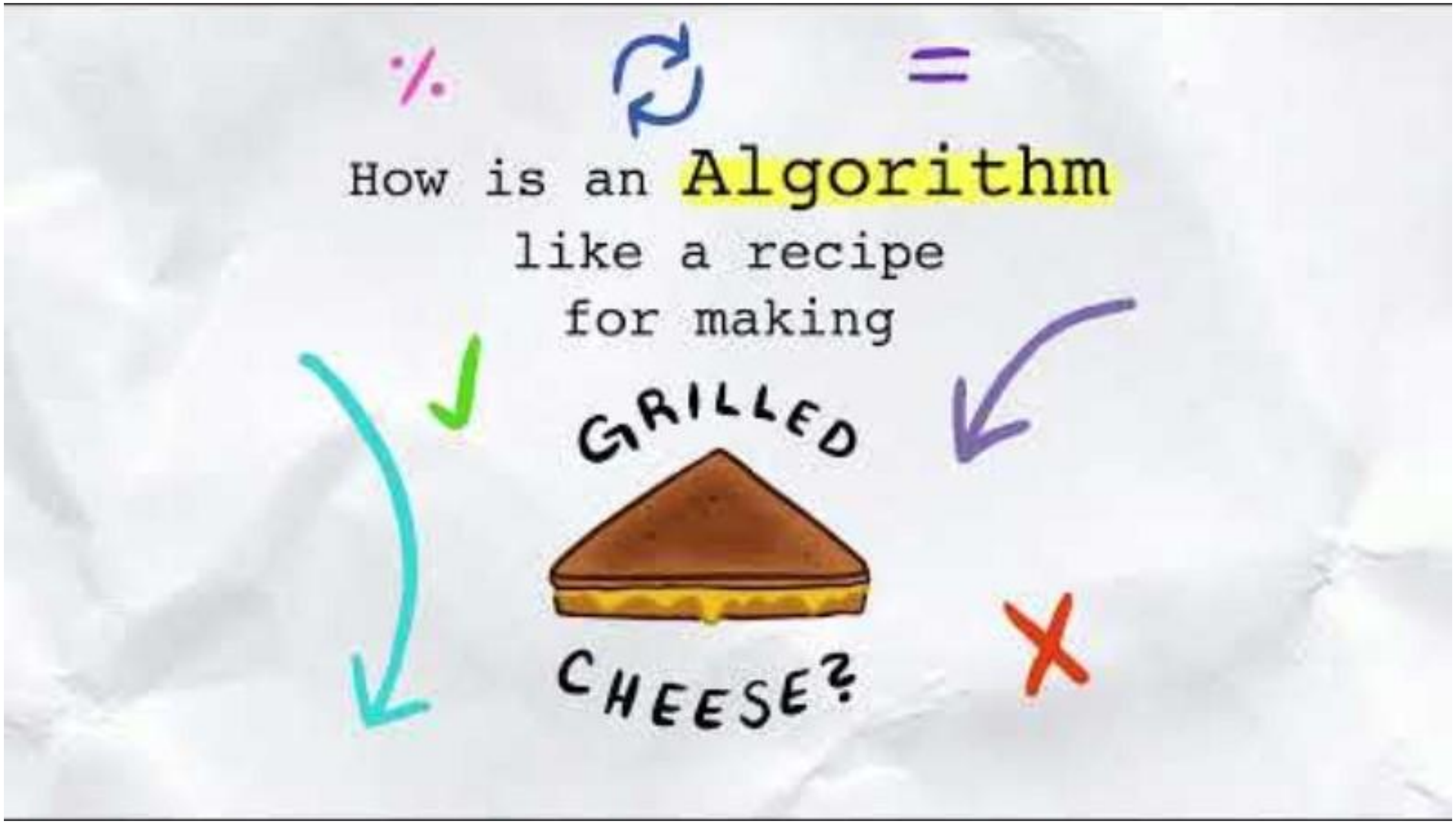
What is an algorithm?

- Simple Definition: **An algorithm is a set of instructions to solve a problem or complete a task.**
  - Think of it like a recipe:
    - if you want to make tea, you follow steps like boiling water, adding tea leaves, and pouring it into a cup. That's an algorithm in action.

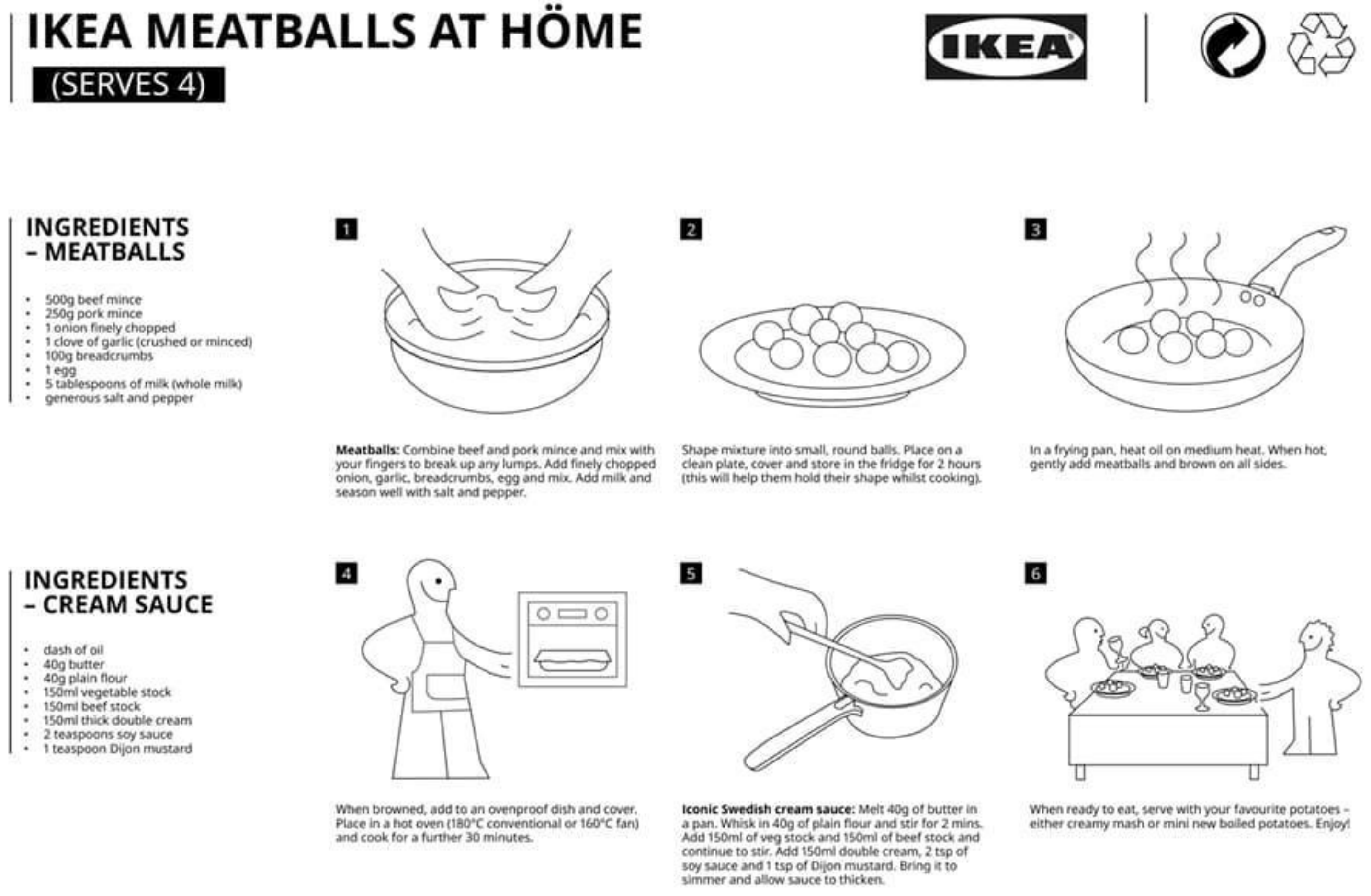
- Intermediate Definition: **An algorithm is a finite, ordered sequence of well-defined steps that take input, process it, and produce output.**
  - This version emphasizes that:
    - The steps must be **clear and unambiguous**
    - The process must **end eventually**
    - It must **solve a specific problem**

- Example:
  - sorting a list of numbers from smallest to largest is a common algorithm used in programming.

Algorithm example (1)



Algorithm example (2)



IKEA MEATBALLS AT HÖME

(SERVES 4)

INGREDIENTS – MEATBALLS

- 500g beef mince
- 250g pork mince
- 1 onion finely chopped
- 1 clove of garlic (crushed or minced)
- 100g breadcrumbs
- 1 egg
- 5 tablespoons of milk (whole milk)
- generous salt and pepper

INGREDIENTS – CREAM SAUCE

- dash of oil
- 40g butter
- 40g plain flour
- 150ml vegetable stock
- 150ml beef stock
- 150ml thick double cream
- 2 teaspoons soy sauce
- 1 teaspoon Dijon mustard

1 **Meatballs:** Combine beef and pork mince and mix with your fingers to break up any lumps. Add finely chopped onion, garlic, breadcrumbs, egg and mix. Add milk and season well with salt and pepper.

2 Shape mixture into small, round balls. Place on a clean plate, cover and store in the fridge for 2 hours (this will help them hold their shape whilst cooking).

3 In a frying pan, heat oil on medium heat. When hot, gently add meatballs and brown on all sides.

4 When browned, add to an ovenproof dish and cover. Place in a hot oven (180°C conventional or 160°C fan) and cook for a further 30 minutes.

5 **Iconic Swedish cream sauce:** Melt 40g of butter in a pan. Whisk in 40g of plain flour and stir for 2 mins. Add 150ml of veg stock and 150ml of beef stock and continue to stir. Add 150ml double cream, 2 tsp of soy sauce and 1 tsp of Dijon mustard. Bring it to simmer and allow sauce to thicken.

6 When ready to eat, serve with your favourite potatoes – either creamy mash or mini new boiled potatoes. Enjoy!

What is an algorithm? (cont.)

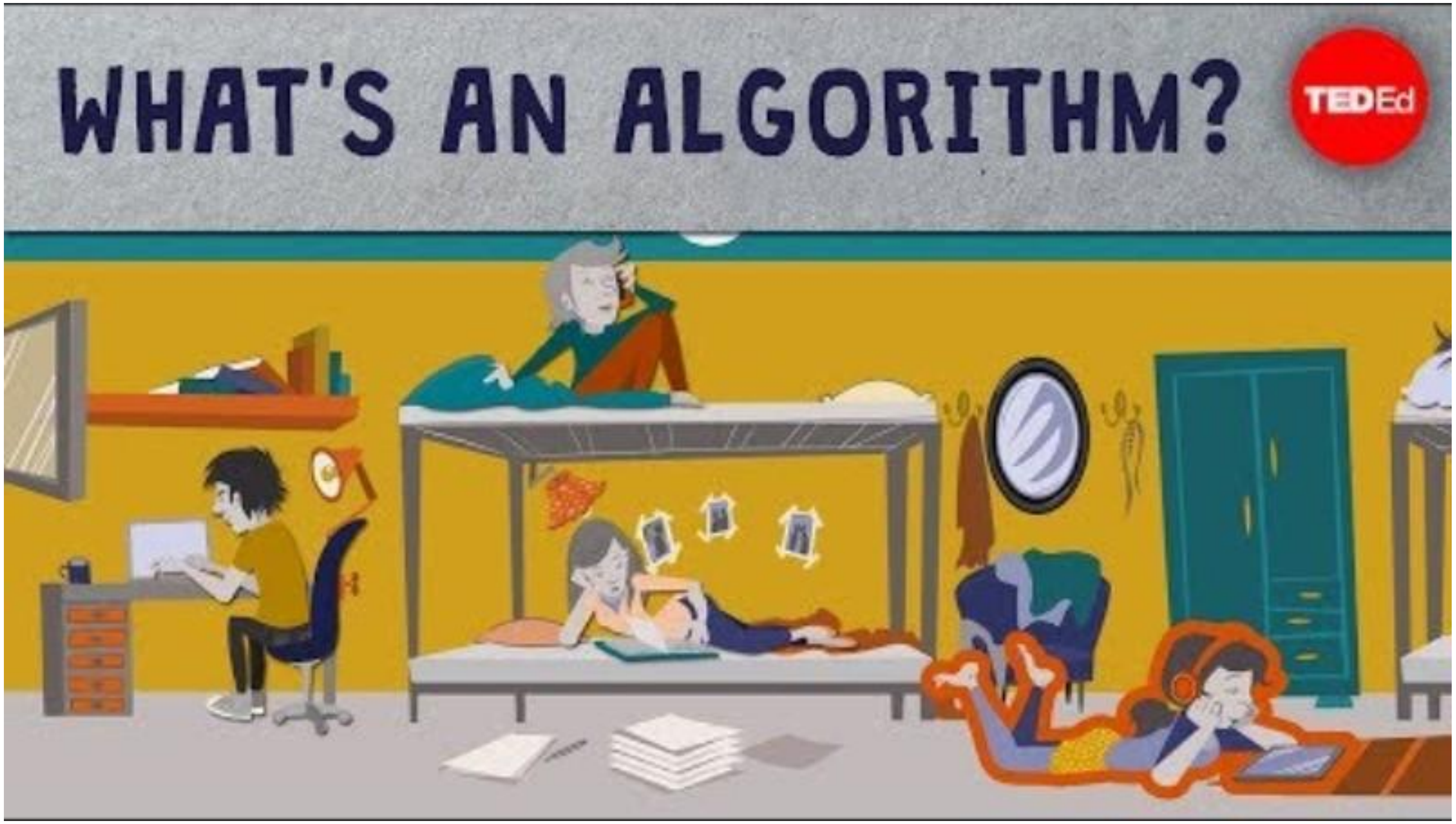
- **Complete Technical Definition:**

  An algorithm is a well-defined computational procedure consisting of a finite sequence of deterministic steps that transform input data into output, designed to solve a class of problems efficiently and correctly.

- This version highlights key properties:
  - **Finiteness**: It must terminate after a limited number of steps
  - **Definiteness**: Each step must be precisely defined
  - **Input/Output**: It may take zero or more inputs and must produce at least one output
  - **Effectiveness**: Each step must be basic enough to be carried out
  - **Correctness**: It must solve the problem as intended

- Algorithms are the backbone of computer science—they power everything from search engines to GPS navigation systems.

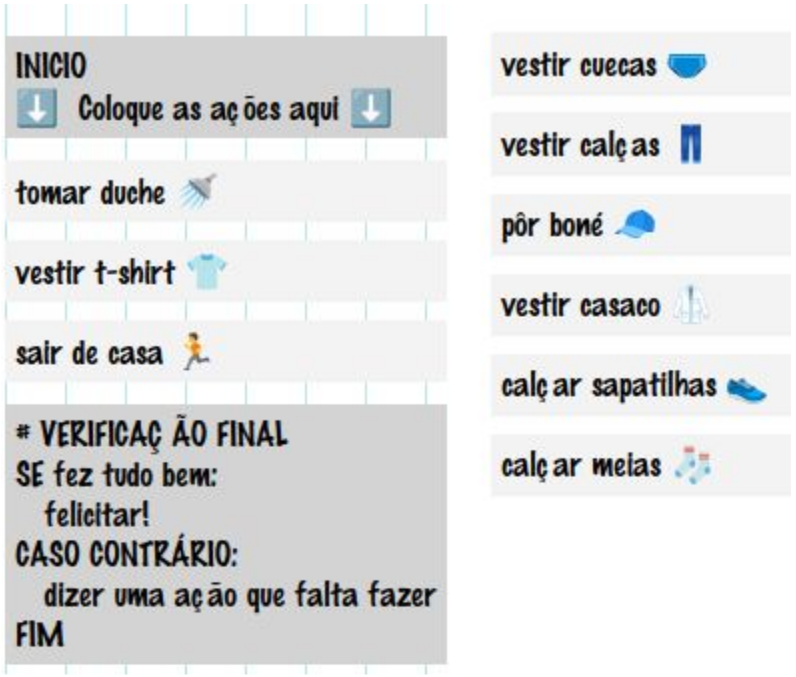
Algorithms



Play ▶️

What is you morning algorithm?

⏰ Acabaste de acordar e tens de te preparar para sair. Qual é o algoritmo que usas?



```
INICIO
⬇️ Coloque as ações aqui ⬇️
tomar duche 🚿
vestir t-shirt 👕
sair de casa 🏃
# VERIFICAÇÃO FINAL
SE fez tudo bem:
   felicitar!
CASO CONTRÁRIO:
   dizer uma ação que falta fazer
FIM
```

vestir cuecas 🩲

vestir calças 👖

pôr boné 🧢

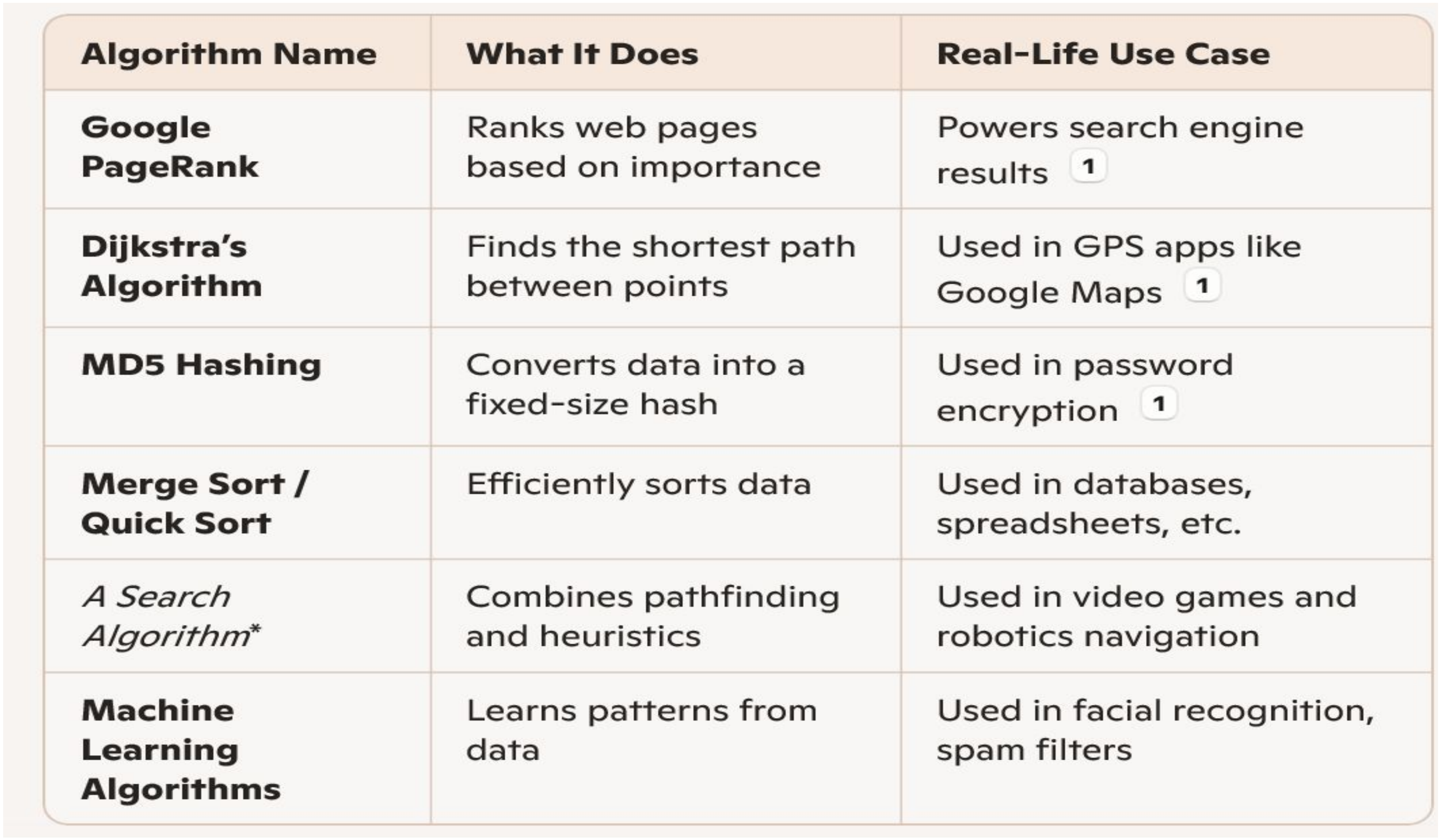
vestir casaco 🥼

calçar sapatilhas 👟

calçar meias 🧦

Try solving it!
(Use a computer.
It does not work well on smartphones, sorry!)

Famous Algorithms in Computer Science



Algorithm Name	What It Does	Real-Life Use Case
Google PageRank	Ranks web pages based on importance	Powers search engine results [1]
Dijkstra's Algorithm	Finds the shortest path between points	Used in GPS apps like Google Maps [1]
MD5 Hashing	Converts data into a fixed-size hash	Used in password encryption [1]
Merge Sort / Quick Sort	Efficiently sorts data	Used in databases, spreadsheets, etc.
A Search Algorithm*	Combines pathfinding and heuristics	Used in video games and robotics navigation
Machine Learning Algorithms	Learns patterns from data	Used in facial recognition, spam filters

Algorithms in everyday life

- Making a cup of coffee:
  - You follow a step-by-step process—boil water, grind beans, brew, pour.
  - That’s an algorithm.
- Choosing what to wear:
  - You check the weather, your mood, and what’s clean.
  - That’s a decision-making algorithm.
- Online shopping recommendations:
  - Algorithms suggest products based on your browsing history.
- Social media feeds:
  - Platforms like Instagram and Facebook use algorithms to show you posts you’re most likely to engage with.
- Navigation apps:
  - Waze or Google Maps use algorithms to find the fastest route and avoid traffic.

Simple algorithm

- Finding the largest number in a list
  - Example: 7, 4, 3, 1, 9, 2, 5

- What to do first?
- What to do next ?

1. Start with the first number as the largest
2. Go through each number in the list
3. If a number is bigger than the current largest, update the largest
4. After checking all numbers, return the largest

Application Examples

ACTIVITY 1: DESCRIBE THE CHANGE OF A TIRE IN A CAR

ACTIVITY 2: DESCRIBE 'MAKE TEA' FOR A ROBOT

Algorithms Require Precision

- Computers need exact steps
- Example: 'make tea' → robot may pour water on the table

- Importance of clarity and order

Problem Decomposition

- Divide & conquer approach
  - Big problems → smaller steps

- Example:

  'Plan a trip' = transport, lodging, activities

- In programming: functions, modules, classes

Abstraction

- Focus on what, not how
- Layers:

  Idea → Code → Interpreter → Machine
- Enables reusability and clarity

Homework

- Write an algorithm for withdrawing money from an ATM
- Do these codecheck exercises.

PROBLEMS

- **Coffee Price Calculator**

  At a coffee shop, each cup of coffee costs €1.50. A customer orders several coffees, and the cashier wants to know how much they need to pay in total. Write a program to help with this.

- **Movie Ticket Counter**

  A cinema sells tickets for €8.00 each. Someone is buying tickets for a group of friends, and they need to know the total cost. Write a program to compute it.

- **Temperature Converter**

  Weather forecasts in some countries are given in Fahrenheit, but in others in Celsius. Write a program that helps convert a temperature given in Celsius to its Fahrenheit equivalent.

Solution (for the first)

- What do you propose as an agile solution for the problem of calculating **Coffee Price ?**

- How to do it?

- Suggestions ?

Possible Algorithm

Algorithm: Coffee Price Calculator

- **Start**
- **Read** the number of coffees the customer wants (from user input).
- **Store** the fixed price of one coffee (€1.50)
- **Multiply** the number of coffees by the price of one coffee to get the total cost.
- **Display** the total cost with a clear message.
- **End**

The complete process

Includes the mental steps you take to obtain the algorithm

- Read & restate the problem
  - Restate in one sentence: A cashier needs the total to charge a customer who ordered some coffees (each €1.50).
- Identify inputs and outputs
  - Input: number of coffees the customer wants.
  - Output: total amount to pay (currency, with two decimals).
- List constraints and assumptions
  - Price per coffee is fixed (€1.50).
  - Number of coffees is a whole number (≥ 0).
  - Handle obvious edge cases (0 coffees, very large numbers, non-numeric input).
- Work a manual example
  - If number = 3 → total = 3 × 1.50 = €4.50.
  - Doing this by hand helps spot rounding/format issues.

The complete process (cont.)

- Find the formula / core idea
  - total = number_of_coffees * price_per_coffee`.
- Choose variables and data types
  - `n` (integer) for number of coffees.
  - `price` (float) = 1.50.
  - `total` (float) for result.
- Decide extra behaviour
  - How to treat invalid input? (e.g., re-ask or show an error)
  - How to format output? (e.g., two decimal places, show € sign)
- Write step-by-step algorithm / pseudocode
  - Produce a straightforward sequence of operations

The complete process (cont.)

- Translate algorithm to code
- Test
  - Try several inputs: 0, 1, 3, 1000, and an invalid input like `-2` or `abc`.
  - Verify output formatting and correctness.
- Refine & document
  - Add input validation, friendly messages.
  - Add comments so you and others understand each step.

Final algorithm (ready to translate into code)

1. Start.
2. Display a prompt asking for the number of coffees.
3. Read the input and convert it to an integer (call it `n`).
4. If `n` is negative or the input is not a valid integer, show an error message (or ask again).
5. Set `price` = 1.50.
6. Compute `total` = `n * price`.
7. Format `total` to two decimal places and display a clear message like: `The total price is €X.XX`.
8. End.

Simple pseudocode

```
PRINT "How many coffees do you want?"
READ input_string

TRY convert input_string to integer n
IF conversion fails OR n < 0:
    PRINT "Invalid number of coffees."
    STOP

price = 1.50
total = n * price

PRINT "The total price is €" + format(total, 2 decimal places)
```

What do we need to know to complete the process?

- What is Python
  - And its specificities
- Display a prompt
- Read the input
- Call it `n`
- …
- Run the program

We'll do this in next class…