45 Programming Languages In 45 Minutes

45 Programming Languages In 45 Minutes

Table of Contents

1. Zoom

- Open a browser to zoom.us
- Click on `Join` at the top
- Enter 388 023 5098
- Launch meeting in browser
- Don't bother with audio ("Continue")
- Watch my show

This file is available as an Emacs Org-mode notebook on GitHub at tinyurl.com/trio-languages.

2. Overview

The purpose of this short session is to show and discuss programming languages with you - not as a theoretical but as a practical thing: I'm going to demonstrate how not one but many languages are used in class so that you have an idea what awaits you at Lyon College if you study computer or data science.

It will emerge that you won't see 45 but only 15-20 languages today but hopefully you'll learn something anyway!

These are some questions I'd like to answer:

1. What is a programming language?
2. How many programming languages are there?
3. What is a programming environment?
4. What is the difference between a programming and a 'natural' language?
5. What is 'natural language processing'?
6. What is Emacs? What is Org-mode?
7. What is Literate Programming?
8. How many programming languages do you 'speak'?
9. Why do you need to know more than one programming language?
10. Which programming languages should you learn (first)?

There will be a short multiple choice quiz at the end!

3. What is a programming language?

A programming language is a *formal language* to express instructions for a computer. It is more like mathematical expressions than like spoken languages.

- Here is a simple example in the C programming language

```
printf("Hello, class!");
```

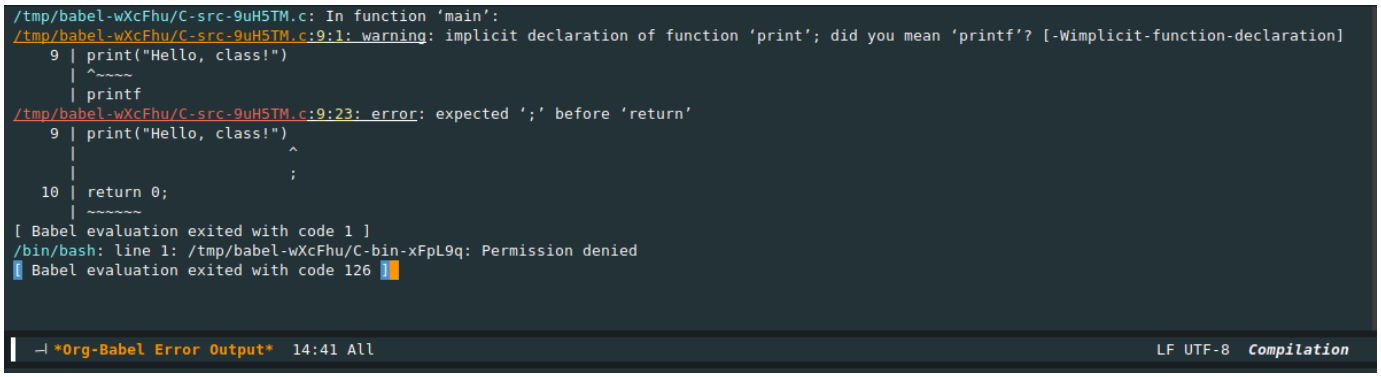
```
Hello, class!
```

- *Formal* in this context means that a programming language requires you to follow its syntax (its set of rules) to the letter. Any deviation, no matter how small, is punished with a "compilation error".
- A *compilation error* means that the computer cannot understand your code (aka *source code*) at all, and cannot use it to derive instructions from it. It will give up talking to you.

- You can see this from the simple program above:

```
print("Hello, class!")
```

- What's different, and what does the error message mean?



- Answer:

  Two errors were highlighted: the command `print` is not recognized (the compiler rightly thinks you meant to write `printf`), and a semi-colon ; is missing at the end of the command.

  We've violated the syntactical rules twice:

  1. we tried to use a function `print` that does not exist
  2. every command in the C language must end with a semi-colon '`;`'

- Philosophical-political extension:

  At GitHub, programming languages are said to "facilitate global conversations between machines and [software] developers, wherever they may be." That's *globalism*[1] for software developers (after all, GitHub is owned by Microsoft).

4. So many technical terms

One of the issues when learning how to program is the sheer amount of new words. In this short introductory section alone we used the following terms:

TERM	EXPLANATION
Emacs	Self-extensible editor
Org-mode notebook	Interactive notebook file in Emacs
C	One of the first programming languages
`printf`	Built-in C function to print to the screen
Formal language	Language with strict rules
Source code	Human-readable code
Machine code	Executable, not readable, code
Compiler	Software to turn source into machine code
Compiler error	
Syntax	Set of rules (grammar)
GitHub	Software development platform
Globalism	World-view that transcends (all) borders

5. What is a programming environment?

- **Tools**: Just like you don't write with your fingers in the open air, but with a pen on paper, or with a keyboard on a screen, 'speaking' a programming language depends on certain tools - the totality of these tools is called "programming environment".
- **Why?** A programming environment is software that allows you to write and run programs on a computer so that you get the same results when you apply the same conditions - this is important because computer programs are usually running unsupervised and cannot check up on themselves, but also because the process of *debugging* (finding program mistakes) relies on repeatable conditions.
- **Files:** The fundamental object of such an environment is the *file* - on today's computers, everything is organized as a file - a part of the computer memory. Therefore it's important to understand how files are created, changed, moved around, and found.
- At Lyon, in my computer and data science classes, I force students to use *Emacs*, the "hacker's editor", to write and run code - this is highly unusual since Emacs is supposed to have too steep a learning curve for beginners (I don't think it does or if it does, it's worth it - cp. Birkenkrahe, 2023).

5.1. Environment example: the shell (aka command line terminal)

- Something you can do on (almost) any computer (except a Chromebook/netbook though even there you can get to it with a little effort) is to open a terminal or a command line.
- Do this now! On Windows: search for `CMD` to find the terminal. (On macOS, search for `terminal`.)
- The terminal opens in a separate window and accepts so-called `shell` commands because it is run via software that sits like an outer layer on top of the *operating system*, the software that controls all processes on a computer.
- For example, to get an overview of the files, you can type `ls` (on Linux and macOS) or `DIR` on Windows. That's the shell command that lists files.
- Shell commands are tiny (with short names) like `ls`, `cd`, `mv`, `rm` for *listing*, *change directory*, *moving*, *removing* files, with many options to alter the command's behavior, and with the possibility of combining commands as part of a command *pipeline*.

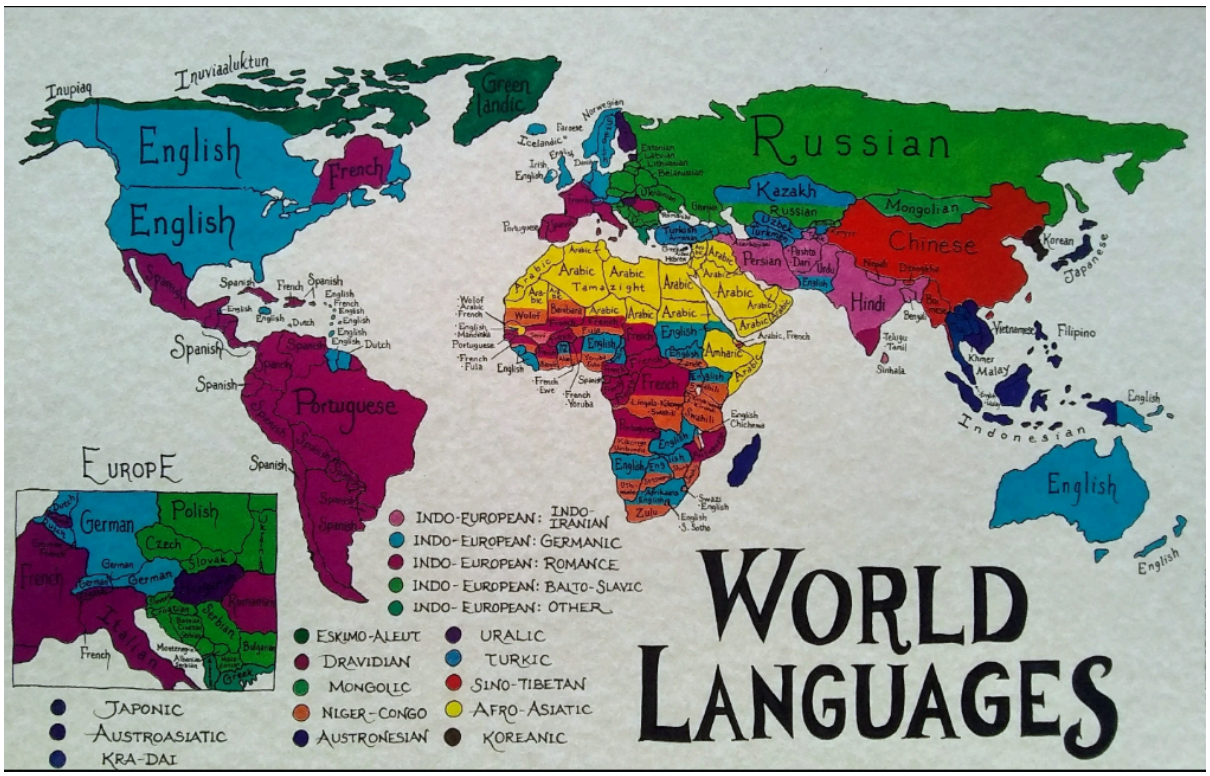
- The code below

  1. lists all files whose names start with 2022
  2. extracts files ending in `.txt`
  3. saves the result to a file named `$2022_{txt}$`
  4. counts the lines of the result (the number of files ending in .txt)

```
ls -l |  grep ipynb | tee ipynb_files | wc -l
```

- You can also program on the shell. The corresponding scripting language is called `bash` ("Bourne-Again-SHell"). I'll show you later.

6. How many [programming] languages are there?



- Spoken ("natural") languages:

  There are more than 7,000 natural (spoken) languages - hard to say because we don't really know what language is ("Language is self-expression" defers the definition to "self"), where it comes from, or when a dialect is a new language (Source: WorldAtlas).

- Programming ("formal") languages:

  There are more than 8,000 historical programming languages of which about 700 are known to be currently used (Source: GitHub).

7. What is the difference between programming and 'natural' language?

- What do you think? How does one approach such a question?

7.1. Approach

Think about different aspects of language:

- Why does language exist?
- How is it structured?
- How is it spoken?
- How is it learnt?
- Which tools do you need to speak/write?
- Where does it come from?
- Do animals have language?
- Can machines speak?

7.2. Answers

- "ChatGPT, contrast programming with natural languages":

Aspect	Programming Languages	Natural Languages
Purpose	To instruct computers to perform specific tasks	To facilitate communication between humans
Structure	Highly structured, with strict syntax and semantics	Less structured, more flexible, context-dependent rules
Precision	Requires high precision, every detail must be correct	Can tolerate ambiguity and imprecision
Ambiguity	No ambiguity, commands must be clear and unambiguous	Often contains ambiguity, resolved by context
Grammar Rules	Strict, formal grammar rules	Complex, often with many exceptions and irregularities
Learning Curve	Steep initial learning curve, but predictable once understood	Variable, influenced by prior language knowledge, exposure
Evolution	Evolves through community consensus and formal proposals	Evolves naturally through usage and cultural changes
Error Handling	Errors are not tolerated; must be debugged	Errors (misunderstandings) tolerated and corrected
Examples	Python, C++, Java, R	English, Spanish, Chinese, Arabic
Tooling	IDEs, compilers, debuggers	Dictionaries, grammar books, conversational practice
Output	Machine-executable programs	Human understanding and action
Dependency	Dependent on hardware and software environment	Dependent on cultural and social context

- **Where does language come from?**

  Perhaps from God? "In the beginning was the Word, and the Word was with God, and the Word was God." John 1:1

- **Do animals have language?**

  Perhaps (not). Like many other beings (even plants) they seem to communicate - but does communication need language?

- **Can ChatGPT speak?**

  It says about itself: "Yes, I can understand and generate natural language. I can process and respond to queries, generate text in various formats, and assist with a wide range of topics using natural language." (The straight answer is NO.)

- **Does ChatGPT really "understand"?**

  Answer: "I don't understand *like a human does*. While I can process and generate text in ways that often seem very human-like, my "understanding" is based on patterns in the data I was trained on, not on conscious thought or human experience. I analyze the input text and generate responses based on patterns, probabilities, and algorithms, but I lack awareness, emotions, and the ability to truly comprehend or experience the world. My responses are designed to be useful and coherent, but they don't come from personal experience or genuine understanding."

  Pretty honest - but does "understanding not like a human" mean anything?

- Summarizing this little detour: without understatement, we can say that the arrival of large language models does not clarify but muddle the issue of language as a concept.

8. What is 'natural language processing'?

I added this because Natural Language Processing (NLP) is the basis of transformer technology, which in turn is behind Large Language Models (LLMs) - software that you know as conversational bots like ChatGPT.

- NLP is one of the most important areas of data science. It involves:
  1. Data collection (gathering digital media - text, images, films, audio)
  2. Data preprocessing (cleaning, tokenizing, normalizing, removing stopwords)
  3. Feature extraction (bag-of-words, term-frequency, Part-of-Speech tagging, etc.)
  4. Model selection (naive Bayes, SVM, transformers)
  5. Model training (split data, train model, test model)
  6. Model evaluation (check accuracy, precision, recall)
  7. Post-processing
  8. Deployment
  9. Hyper-parameter tuning
  10. Maintenance

8.1. Two NLP examples: Bag-of-words and Syntactic parsing

These examples come from my freshman course on text mining - (CSC 105 Digital Humanities):

8.1.1. Bag of Words - simple example

- Bag of Words generates a document-term matrix (DTM) or their transposition, a term-document matrix (TDM).
- In a DTM, each row represents a document or individual *corpus*, e.g. a tweet, and each column represents a word. In a TDM, rows and columns are switched.

- Example: three tweets form a *corpus* or body of text for analysis. These are tweets about the statistical programming language R (hashtag `#rstats`):

  - **@hadleywickham:** "How do I hate thee stringsAsFactors=TRUE? Let me count the ways #rstats"
  - **@recodavid:** "R the 6th most popular programming language in 2015 IEEE rankings #rstats"
  - **@dtchimp:** "I wrote an #rstats script to download, prep, and merge @ACLEDINFO's historical and realtime data."

- A document term matrix (DTM) for this corpus:

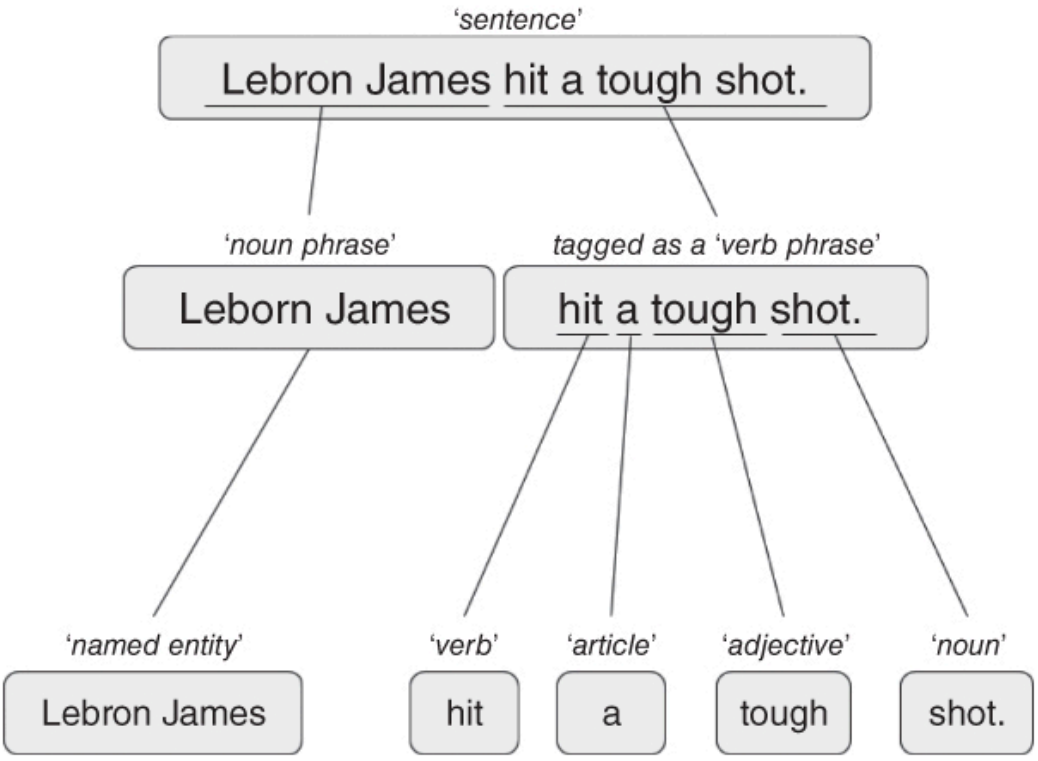
Tweet	@acledinfo's	#rstats	2015	6th	And	Count	Data	Download	...
1	0	1	0	0	0	1	0	0	...
2	0	1	1	1	0	0	0	0	...
3	1	1	0	0	2	0	1	1	...

- The transposed DTM or transposed document matrix (TDM):

Word	Tweet1	Tweet2	Tweet3
@acledinfo's	0	0	1
#rstats	1	1	1
2015	0	1	0
6th	0	1	0
And	0	0	2
Count	1	0	0
Data	0	0	1
Download	0	0	1
...	...	...	...

- These DTM and TDM examples only show word counts. Now, without reading all the tweets (perhaps a much larger number than three), you can surmise that the tweets are related to R.
- You can also see that there are some words like 'data', 'download', or 'and', that are very common and won't add to the analysis: these can be removed using so-called 'stopword' dictionaries.

8.1.2. Syntactic parsing - simple example



- *Syntactic* or *semantic* parsing has many more attributes assigned to a sentence than Bag-of-Words; it captures & retains more information.
- Syntactic parsing involves determining the roles that each word plays in a sentence (e.g. noun, verb, adjective, etc.) and their relations.
- It is often used as a first step in natural language processing (NLP), before more advanced analysis can be applied.
- Semantic parsing is the process of interpreting natural language input and determining its meaning.
- To do that, sentences have to be mapped to a representation, e.g. by tagging Parts-of-Speech (POS) as building blocks.
- Tags are captured as *meta-data* of the original sentence.

9. Summary I

- A **programming language** is a formal language to express instructions for a computer.
- Programming requires an **infrastructure** including an environment with tools like an editor, a compiler, and a shell
- You use an **editor** to create source code, a **compiler** to generate machine code, and a **shell** to interact with the computer's OS
- **Natural languages** are a complete mystery but they can be processed to build **generative AI** tools that can augment human problem-solving
- **Bag-of-words** and **syntactic parsing** are two common Natural Language Processing (NLP) methods.

10. What is Emacs? What is Org-mode?

We've talked about programming and natural language and about the environment needed to process natural language using programming. Now we're going to talk about the most powerful programming environment available to man - and it exists since 40 years!

10.1. Emacs - Editor MACroS (1985)

- Emacs is a highly customizable **editor**. **VSCode** (Visual Studio Code) uses a similar plugin approach to enhance its capabilities.
- The difference: Emacs is totally **Free and Open Source Software** (FOSS), and that it can be changed by anyone who knows Emacs Lisp.
- Emacs is often called the "hacker's editor" because it is so extensible, integrates with so many tools, and is capable of handling so many tasks around software development with ease.
- To find out more, watch my Emacs tutorial I and II (30 min), and look at my playlist for students (tinyurl.com/trio-playlist). If you come to Lyon to study with me, you have to learn Emacs anyway.
- If someone says "Emacs is hard to learn", don't believe them, just like when someone says "Linux is harder than Windows or macOS". The opposite is the truth.
- Most importantly for you as a learner, everything you do when you learn Emacs has **high transfer value**, i.e. you can use it somewhere else and it helps you understand more of what you need to know.

10.2. Org-mode

- Org-mode is a structured plain text file format, and an Emacs package for organizing, authoring, and managing notes, tasks and documents within Emacs, but also to program without having to ever change the programming environment.
- This whole presentation is written in Org-mode and presented in Emacs using the `org-tree-slide` package.

- What you see right now, is not **WYSIWYG** (What You See Is What You Get) but **WYSIWYM** (What You See Is What You Mean):

Aspect	WYSIWYG (What You See Is What You Get)	WYSIWYM (What You See Is What You Mean)
Emphasis	Final appearance	Structure and meaning
Editing	Visual, direct layout manipulation	Abstract, focuses on content/structure
Examples	Microsoft Word, Google Docs	LaTeX, Markdown, Org-mode
Output	Immediate editing reflects final output	Rendering defines final output

- Look at this page vs. a Google Docs page: here, *meta data* elements control the layout. In Emacs + Org-mode the control is mediated through keyboard shortcuts - much faster than the graphical user interface (GUI).
- Here is a copy of the meta data at the top of this file:

```
:PROPERTIES:
:ID:       4e81265d-efb8-4786-89f6-a98b3ccf81b8
:ROAM_ALIASES: trio-languages
:END:
#+TITLE:45 Programming Languages In 45 Minutes
#+AUTHOR:45 Programming Languages In 45 Minutes
#+SUBTITLE:45 Programming Languages In 45 Minutes
#+STARTUP:overview hideblocks indent
#+OPTIONS: toc:nil num:nil ^:nil
#+PROPERTY: header-args:R :session *R* :results output :exports both :noweb yes
#+PROPERTY: header-args:python :session *Python* :results output :exports both :noweb yes
#+PROPERTY: header-args:C :main yes :includes <stdio.h> :results output :exports both :noweb yes
#+PROPERTY: header-args:C++ :main yes :includes <iostream> :results output :exports both :noweb yes
```

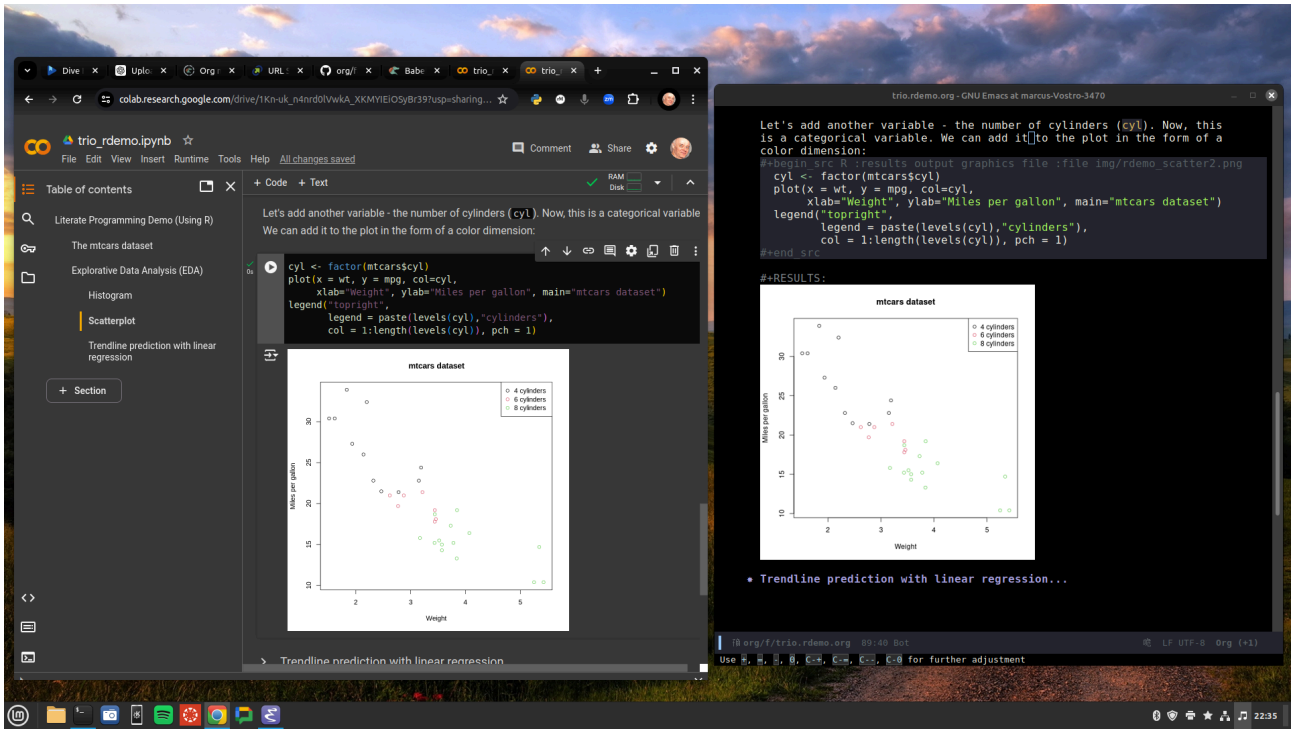
  1. The top (green) is meta data for Org-Roam, a system for notekeeping.
  2. The middle contains title, author, subtitle, layout, rendering info
  3. The bottom instructs Emacs how to run code in R, Python, C and C++
- Org-mode was developed by the German astrophysicist Carsten Dominik: in computer and data science, people from outside the field often develop the most useful tools starting with their own needs.
- To find out more about Org-mode, check out orgmode.org.

11. What is 'Literate Programming'?

- This is a common journey in computer science: start with a problem (programming), develop tools (Emacs + Org-mode), and then organize your thoughts and your workflow into a new "groove".
- This new groove (well, it was new in the 1980s) is "literate programming". It emphasizes writing code that humans can read and understand[2].
- It took the development of interpreted languages like Python and R, and the need of data scientists to explore data in real time to resurrect literate programming in the form of "interactive notebooks".
- These notebooks are partly responsible for the success of data science especially in the form of machine learning and predictive models (like generative AI - ChatGPT as an example).
- The best known literate programming notebook is called "Jupyter" for three popular data science languages, Julia, Python, and R. Here is an example - which we're going to use in the other 2 workshops.
- You can see that the notebook contains text + code + output. In this case, I'm looking at a built-in car data set. Check out the link to run the code cells yourself (you may need a GMail acct): tinyurl.com/trio-rdemo-colab[3]
- Jupyter notebooks (and their derivates) are pretty bloated and they can only run one programming language at a time[4], and you cannot directly use not interpreted languages like C,
- Emacs + Org-mode however can run 45 programming languages in one document. Why is that helpful? Because different languages can do different things.

11.1. Compare Emacs + Org-mode with Colab

- Contrast the Emacs + Org-mode original (available on GitHub) with the Colab version (available on Google Colaboratory):



- You can see the difference in complexity of the GUI: on the left hand side (Colab), you see the usual menu structure that you're used to from WORD or GoogleDocs (WYSIWIG), while on the right hand side all layout and functionality is contained in the (textual) metadata.
- I am currently very motivated to speak about literate programming, because I've been asked to write a book for a publisher on this topic, and I have even received a research grant to do this.
- Take a closer look at Google Colab and a literate program - a demonstration using the programming language R: tinyurl.com/trio-rdemo-colab - open the file (you may have to register or login with a Google account) and run the cells.

12. How many programming languages do you 'speak'?

Incidentally, these are also the 16 programming languages that you can learn when you take computer or data science classes at Lyon College.

I know a few more programming languages (20) but I don't use them on a regular basis. Like natural languages, if you don't use a programming language for a while, you begin to forget it - but it never goes away altogether[5].

The classic program to write as a first program in any language is **"Hello world"**[6]. It's instructive and it's good luck. I'm going to sketch solutions to this program in a few languages below.

I could keep going: Org-mode supports 71 programming languages right out of the box (of which 44 are core languages, with 27 supported, sometimes fairly obscure languages) - see Babel documentation.

12.1. R

R is an interpreted, FOSS, statistical programming language, very strong on visualization and statistical functions. To learn it, check out Matloff's online tutorial fasterR @ GitHub[7].

```
"Hello, world!"
```

```
[1] "Hello, world!"
```

What's really happening here:

```
s <- "Hello, world!" # a one-element character vector
s
is.character(s)
is.vector(s)
```

```
[1] "Hello, world!"
[1] TRUE
[1] TRUE
```

12.2. SQLite/SQL

SQL (Structured Query Language) is the de-facto standard for relational databases, or data repositories where all data are stored as tables. SQLite is a light-weight version of SQL without user management, written in C, with a totally different architecture.

In the example, we use an SQLite database `test.db`

```
SELECT "Hello, world!"
```

```
Hello, world!
```

This is pretty boring and doesn't really use SQL's abilities. Here is a more elaborate and more interesting example:

```
CREATE TABLE IF NOT EXISTS test (INT id, greeting TEXT);
.tables
```

```
test
```

Now we have a table. Let's put the string into it:

```
INSERT INTO test (greeting) VALUES ("Hello, world!");
```

Finally we're ready for the output:

```
SELECT greeting FROM test;
```

```
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
```

SQLite is easy to learn. SQL and/or SQLite are must-have languages for anybody in data science and/or web development.

12.3. Python

Next to R, Python (also written mostly in C) is the other popular language for machine learning. It is also very popular in its own right - as a scripting, interpreted language for beginners that can be used to automate processes, develop games, and scientific computing.

Again, I present a simple and a fancy version of "Hello, world!":

```
print("Hello, world!")
```

```
Hello, world!
```

Or you can store the string in a variable and print it using an f-string:

```
greeting = "Hello, world!"
print(f"{greeting}")
```

```
Hello, world!
```

Or as part of a list:

```
hw = ["Hello", ",", "world", "!", " "]
print(hw[0] + hw[1] + hw[-1] + hw[2] + hw[3])
```

```
Hello, world!
```

Python is famous for the variety of its data structures. And this is only the pale beginning! You can program object-oriented in Python, too.

When you study data science at Lyon College, R + Python + SQL is going to be your staple diet in many courses.

12.4. C/C++

C is the mother of many high-level programming languages, i.e. languages that provide a level of abstraction so that you can express complicated problems almost in natural language - compared to languages like Assembler that are closer to the machine.

For example, here is "Hello world" in Assembly language (for a Linux system):

```
section .data
    hello db 'Hello, World!',0    ; The string to be printed

section .bss

section .text
    global _start

_start:
    ; Write the string to stdout
    mov eax, 4            ; syscall number for sys_write
    mov ebx, 1            ; file descriptor 1 is stdout
    mov ecx, hello        ; pointer to the string
    mov edx, 13           ; length of the string
    int 0x80              ; call kernel

    ; Exit the program
    mov eax, 1            ; syscall number for sys_exit
    xor ebx, ebx          ; exit code 0
    int 0x80              ; call kernel
```

However, C is not that far away from the machine. It allows you to program directly in the machine's memory.

Once again, the simple version of a C Hello World program:

```
#include <stdio.h>

int main()
{
  puts("Hello, world");
  return 0;
}
```

```
Hello, world
```

Unlike R, Python, SQL, it is a compiled language that is there is no way to program interactively. With Emacs + Org-mode, however, this can be achieved, as you see here:

```
puts("Hello, world!");
```

```
Hello, world!
```

Here is a version of "Hello world" in C that uses memory allocation and pointers (references to memory locations):

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
  // Allocate memory for the string
  char *hello = (char *)malloc(14 * sizeof(char));
  if (hello == NULL) {
    fprintf(stderr, "Memory allocation failed\n");
    return 1;
  }

  // Copy the "Hello, World!" string into the allocated memory
  strcpy(hello, "Hello, World!");

  // Print the string
  printf("%s\n", hello);

  // Free the allocated memory
  free(hello);

  return 0;
}
```

```
Hello, World!
```

C++ is an object-oriented extension of C. This simple version looks very similar:

```
#include <iostream>

using namespace std;

int main()
{
  cout << "Hello, world!" << endl;
  return 0;
}
```

```
Hello, world!
```

For the fancy version, let's demonstrate object-oriented programming:

```
#include <iostream>
#include <string>

// Define a class named Greeter
class Greeter {
public:
  // Constructor that initializes the message
  Greeter(const std::string& msg) : message(msg) {}
```

```
  // Member function to print the message
  void greet() const {
    std::cout << message << std::endl;
  }

private:
  std::string message;  // Data member to hold the greeting message
};

int main() {
  // Create an object of the Greeter class
  Greeter greeter("Hello, World!");

  // Call the greet member function
  greeter.greet();

  return 0;
}
```

```
Hello, World!
```

At Lyon College, "Introduction to programming with C/C++" and "Data structures with C++" are the first two programming courses you'll complete for a major or minor in computer science.

12.5. Java

A single introductory programming class will not turn you into a competent programmer - two classes won't either - but an introductory class in Java will help. Also, Java is a widely spoken language for industrial applications. This is because it is highly portable - it comes with its own environment and runs everywhere, and fast.

I'm not very good at Java, so here's only the simple version -

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

12.6. bash

`bash` is a shell script language. It allows you to interact directly with the operating system using the command line.

The minimal hello world program in bash:

```
echo "Hello, world!"
```

12.7. Emacs Lisp

Lisp ("List processing") is an old language - the first language used to code early Artificial Intelligence (AI) applications! A dialect of Lisp is also used to code the Emacs editor and all of its thousands of extension packages (like Org-mode).

Hello world is very simple.

```
(message "Hello, world!")
```

Here is a fancier version that uses Lisp's list processing abilities - the output is printed in Emacs' minibuffer only.

```
;; Define a list of greetings
(setq greetings '("Hello, World!"))

;; Function to print each greeting
```

```
(defun print-greetings (greeting-list)
  "Print each greeting in the list."
  (dolist (greeting greeting-list)
    (message greeting)
    (sleep-for 1)))  ;; Add a 1-second delay between each message

;; Call the function with the greetings list
(print-greetings greetings)
```

12.8. JavaScript

JavaScript dominates the web development world for front-end application programming - e.g. to make web pages dynamic and fun. At Lyon, you learn it as part of an asynchronous, online web development course. It's a language you can pick up easily yourself though, e.g. by programming a few simple games (check out freeCodeCamp).

I cannot run this program in

```
console.log("Hello, World!");
```

12.9. Markdown

Markdown is a lightweight markup language with plain text formatting syntax designed to be easy to read and write, and can be converted to HTML. It's not a proper programming but a layout language only.

As a demonstration, check out the raw version of this README.md file by clicking on the `Raw` tab.

As for "Hello, world!", all you need is the text.

12.10. LaTeX

$LaTeX$ is a high-quality typesetting system used primarily for technical and scientific documents, known for its powerful handling of formulas and bibliographies. It's a set of macros for TeX, a large document publishing package. LaTeX is the common format for scientific papers and publications. For example, I will prepare the manuscript for the book on Literate Programming that I'm writing in LaTeX, and not in WORD.

Here is an example for a complex formula in LaTeX. To see it rendered e.g. in a browser, I use Emacs' Org-mode dispatch mode:

> The Euler-Lagrange equation is given by:
>
> $$\frac{\partial L}{\partial q_i} - \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) = 0$$
>
> where L is the Lagrangian, q_i are the generalized coordinates, and $\dot{q}_i$ are the generalized velocities.

12.11. HTML

HTML (Hypertext Markup Language) is the standard language used to create and design documents on the World Wide Web, defining the structure and layout of web pages.

"Hello, world!" in HTML is just text whose format can be altered with tags, e.g. `<b>"Hello, world!</b>` for bold face etc.

The HTML file from this section looks like this:

```
<p>
HTML (Hypertext Markup Language) is the standard language used to
create and design documents on the World Wide Web, defining the
structure and layout of web pages.
</p>

<p>
"Hello, world!" in HTML is just text whose format can be altered with
tags, e.g. <code>&lt;b&gt;"Hello, world!&lt;/b&gt;</code> for bold face etc.
```

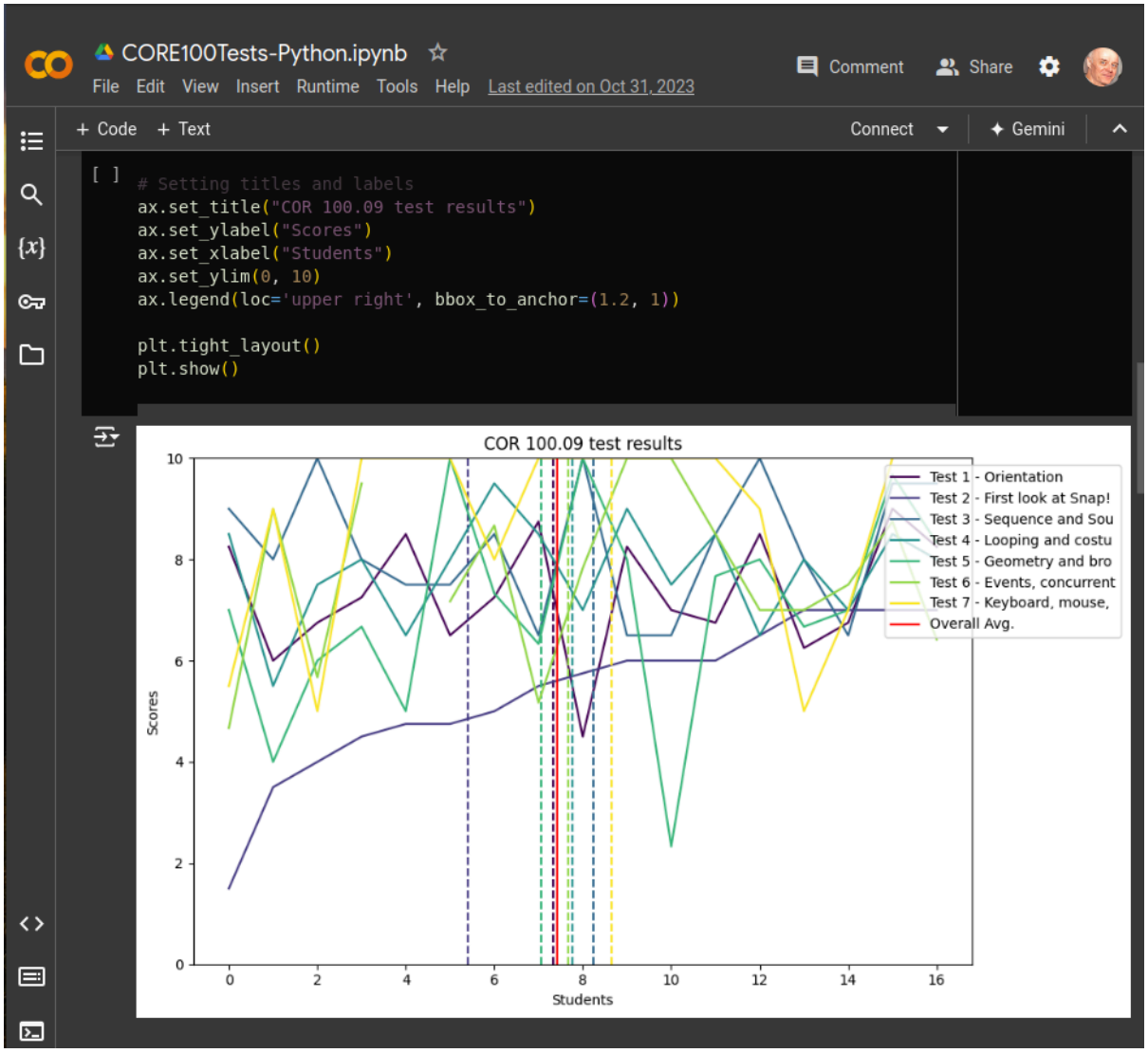
```
</p>
</div>
```

You can render Org-mode files as HTML files for browser display. Once in the browser, right-click to `Inspect` the HTML code.

12.12. IPython

IPython(Interactive Python) is an enhanced interactive shell for Python that provides a rich toolkit to help you make the most of using Python interactively, offering features such as easy access to shell commands, improved introspection, rich media output, and integration with visualization libraries.

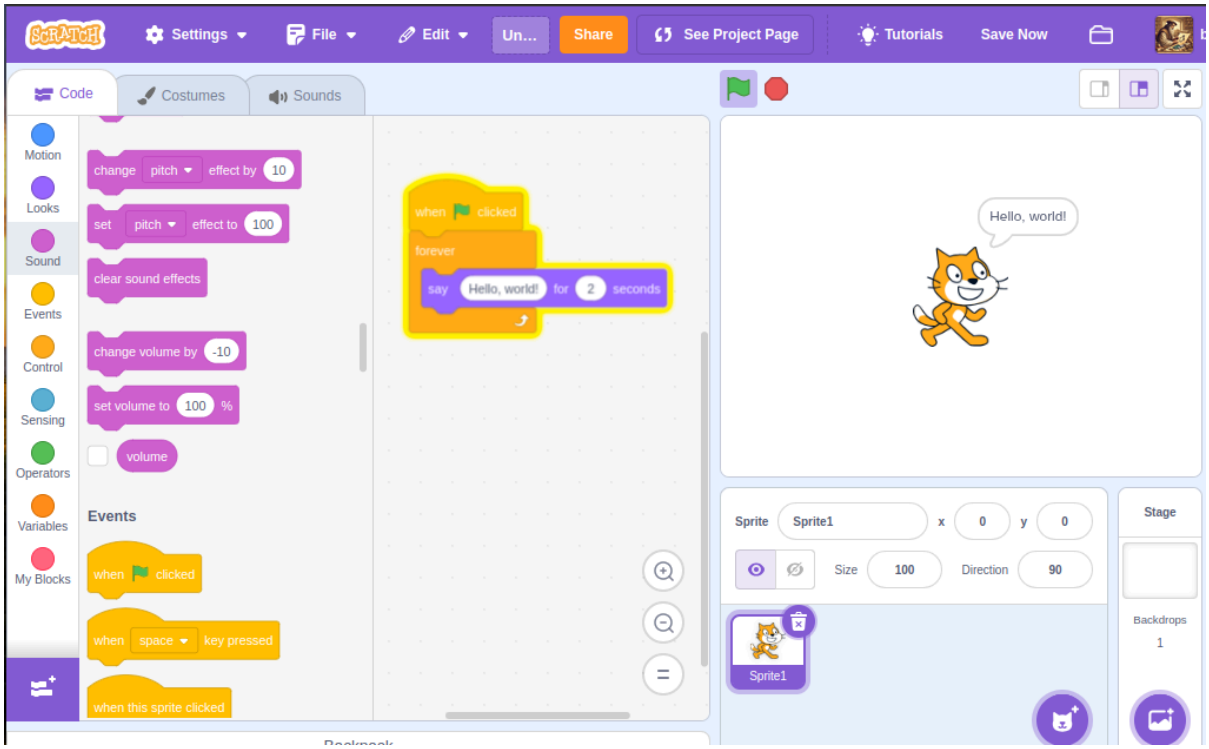
You can see it in action in Google Colaboratory. You've already seen interactive R, and here is a Python IPython notebook (an analysis of test grades from one of my courses).



12.13. Scratch/Snap!

Several of you will remember Snap! from the 2023 summer school where we used this visual, block-based language to program simple animations and 2D games. Scratch, its simpler ancestor is very popular in programming education for kids.

Here is a screenshot of the Hello world program in Scratch:



To find out more, go directly to scratch.mit.edu or snap.berkeley.edu.

13. Why do you need to know more than one programming language?

- Because languages are created easily often for a specific purpose, and today's top language may be superseded by others.
- Many applications are tied to specific languages, and most work places have multiple applications.
- Because as a programmer you want to be a Swiss army knife, and not just one type of weapon for the solution of one problem.
- Not only applications, but machines also change all the time, and to get the most mileage out of them, languages are tweaked.
- Because it's fun to pick up, learn, and flex new languages.

14. Which programming languages should you learn (first)?

This is a very personal question, with a personal answer. It depends quite a bit on what you want to do with programming languages.

I recommend C (which is why I teach it at Lyon as an introductory language) because it's small, powerful, and common, and keeps you focused on the machine. It's not very abstract at all.

Once you've learnt C, you probably want to learn C++ - especially if you're into game programming or cybersecurity or graphics.

If your interest is in data, data science and its applications like machine learning and AI, then Python is your go-to language.

To learn something completely different that will give you enormous power over your work environment, learn Emacs and learn Emacs Lisp.

My own path was different:

- I learnt BASIC because it came with the computer I was given as a teenager.
- I learnt FORTRAN because that's what we had to learn as physics students.
- I learnt REXX because I had a job at a computer center.
- I learnt HTML because the web had just been created and I was in charge of a bunch of virtual libraries (for literate programming, C++, and numerical multigrid methods).
- I learnt C++ because I needed it for my PhD.

15. Bonus: Why do I need to learn any programming language? (What about AI?)

I asked AI, and the answer is pretty good (I added two at the end):

1. Understanding and Control
   - Deeper understanding of how software and systems work.
   - Customization and control over software behavior.
2. Problem-Solving Skills
   - Teaches critical thinking and problem-solving skills.
   - Enables breaking down complex problems into manageable parts.
3. Automation and Efficiency
   - Automate repetitive tasks to increase productivity.
   - Create scripts and tools tailored to specific needs.
4. Career Opportunities
   - High-paying jobs in technology, data science, engineering, and related fields.
   - Competitive edge in the job market.
5. Interdisciplinary Applications
   - Used in various fields such as finance, healthcare, education, and more.
   - Analyze data, create models, and develop industry-specific applications.
6. Innovation and Creativity
   - Build your own projects, from simple applications to complex systems.
   - Fosters creativity and innovation.
7. Interfacing with AI
   - Crucial for working with AI technologies.
   - Develop, train, and fine-tune AI models, and integrate them into applications.
8. Ethical and Responsible Use
   - Understand the ethical implications of technology and AI.
   - Make informed decisions about data privacy, security, and societal impact.
9. Because it's fun!
10. Because you need to know what the machines are doing!

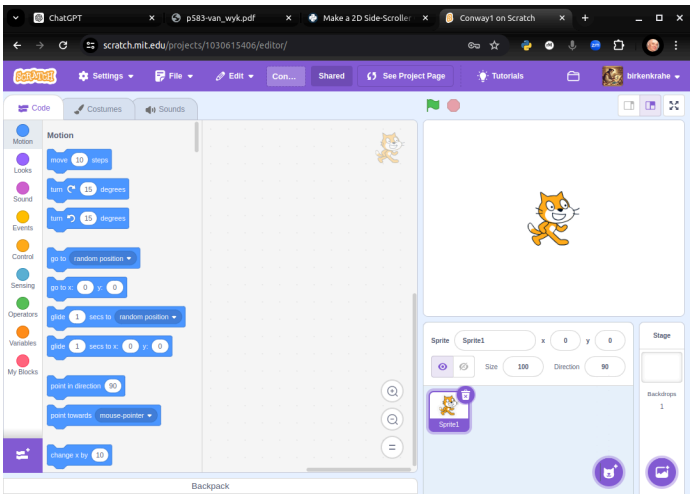
16. Summary II

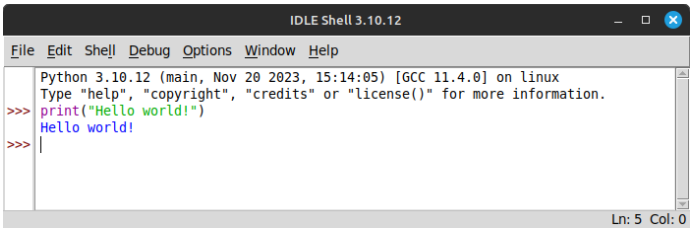
- Emacs is a highly customizable, free and open source editor that trains many transferable skills and will speed you up enormously.
- Org-mode is a structured plain text file format and Emacs package for organizing, authoring and managing notes, tasks, documents, and for developing code and writing literate programs.
- Literate programming emphasises writing code that is human-readable and understandable, is ubiquitous in data science (via Jupyter notebooks), and especially powerful with Emacs + Org-mode.
- It is important to know multiple languages for flexibility, specific applications, and problem-solving.
- Examples include R, SQL, Python, C, C++, Java, bash, Emacs Lisp, JavaScript, Markdown, LaTeX, HTML, IPython, and Scratch.

17. Thank you for your attention!

18. Bonus: What's the difference between Scratch and Python?

- What does it look like?





- Answer:

  Scratch is a high-level visual, block-based, script language: Scratch commands are organized as scripts which need to be run, and you don't have to know the precise form of commands because they're hardcoded as blocks. The programming is done in a script area, and the output is shown on a stage using "sprite" objects (like the cat).

  Python is a high-level, interpreted/compiled programming language: you can run it in its interpreted form from a console, or you can run Python scripts (like Scratch scripts). It is used for teaching how to program but it is also used in professional settings. It is an important language for data science and machine learning.

  Both Scratch and Python are dynamically typed: the data type of variables is not declared or checked at compile time (when the executable program is created). It assigns values to variables and binds them to memory containers that can store any type of data.

  Both languages are considered easy to learn for beginners.

- TIOBE Index June 2024:

May 2024	May 2023	Change	Programming Language	Ratings	Change
1	1		Python	16.33%	+2.88%
2	2		C	9.98%	-3.37%
3	4	↑	C++	9.53%	-2.43%
4	3	↓	Java	8.69%	-3.53%
5	5		C#	6.49%	-0.94%
6	7	↑	JavaScript	3.01%	+0.57%
7	6	↓	Visual Basic	2.01%	-1.83%
8	12	⇈	Go	1.60%	+0.61%
9	9		SQL	1.44%	-0.03%
10	19	⇈	Fortran	1.24%	+0.46%
11	11		Delphi/Object Pascal	1.24%	+0.23%
12	10	↓	Assembly language	1.07%	-0.13%
13	18	⇈	Ruby	1.06%	+0.26%
14	15	↑	MATLAB	1.06%	+0.18%
15	14	↓	Swift	1.01%	+0.09%
16	8	⇊	PHP	0.97%	-0.62%
17	13	⇊	Scratch	0.93%	-0.02%

19. References

- Birkenkrahe (2023). Teaching Data Science with Literate Programming Tools. URL
- GitHub (2024). What is a programming language? URL
- WorldAtlas (2024). How Many Languages Are There In The World? URL

Footnotes:

[1] Globalism has multiple meanings but I associate the term mainly with a world-view that transcends national borders and cultural boundaries. Some jobs are more 'global' than others - anything related to the digital world that is transacted over the Internet for

example. Though when you look at the details, local conditions are not unimportant - e.g. even when you develop web sites, you need to think of your customers as being members of a particular culture, speaking a particular language, etc.

[2] Literate programming is actually a computing paradigm - a new approach that eliminates old ways of doing things in order to deal with "anomalies" (an unexpected issue) of programming: in this case the difficulties of very large software projects. Another structural paradigm that was introduced to address the same issue is Object-Oriented Programming (OOP).

[3] This Google Colaboratory notebook was rendered from the original Emacs + Org-mode notebook at tinyurl.com/trio-colab-demo-org. While the Org-mode file can be rendered in any text-based format, the Jupyter notebook format (.ipynb) is what it is and it is tied to the application (that's a loss of flexibility).

[4] Each notebook runs on a so-called language kernel for either R or Python. The kernel in turn sits in a Linux container, a virtualized operating system - that's the notebook's *runtime environment*. You cannot switch language, e.g. to use R for quick visualization, without switching to a new notebook. The only exception is SQL - notebooks can draw on relational databases and SQL commands.

[5] Other languages I know (but don't use): BASIC, FORTRAN (those were my two first languages), Pascal, and REXX (which I used when working on an IBM mainframe at DESY).

[6] This program goes back to Kernighan & Ritchie's first C programming book "The C programming language", which was the first proper programming book in 1978, and is still a brilliant, short, very dense book to work through (there's a 2nd edition, 1988).

[7] Incidentally, R was my first proper "data science" language. I learnt it alongside learning data science methods in 2019.

Author: 45 Programming Languages In 45 Minutes
Created: 2024-06-10 Mon 14:33
Validate