INF1-IP

Introduction to Programming

Lecture 15 – Programming languages

Matteo Degiacomi

THE UNIVERSITY of EDINBURGH
informatics

> Why so many languages?

Hundreds of programming languages have been created. **Why?**

Today's promise: you will read a language nobody has taught you.

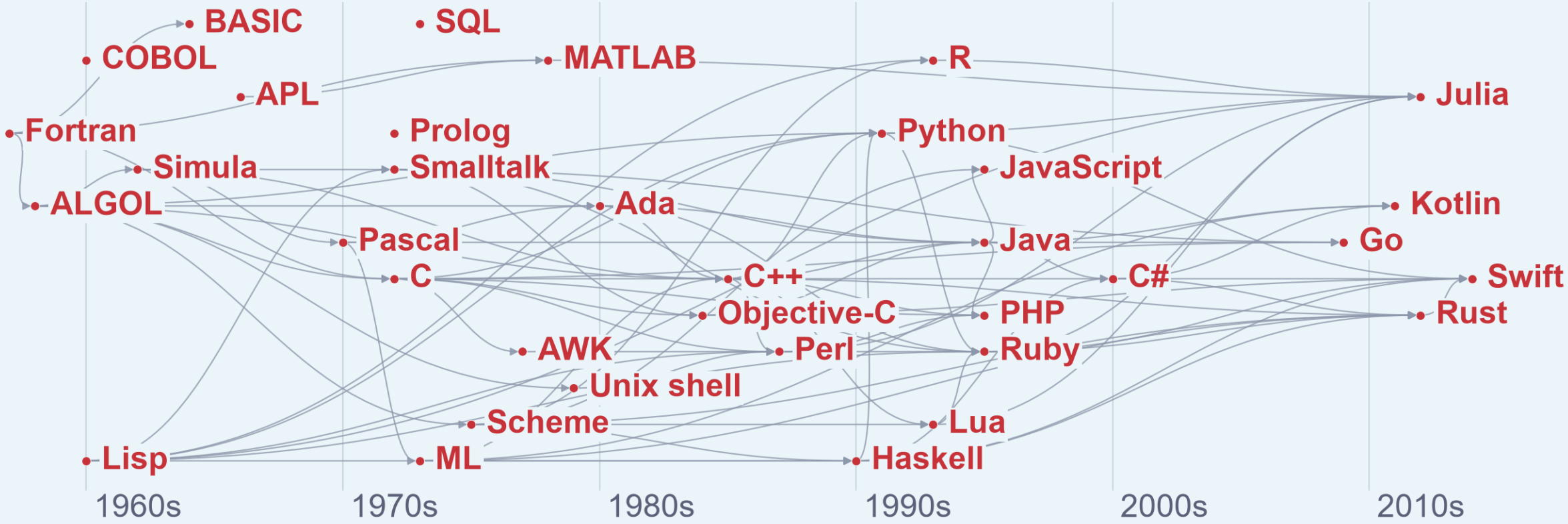
What counts is giving a **reason**.



The Great Tower of Babel
Pieter Bruegel the Elder, c.1563

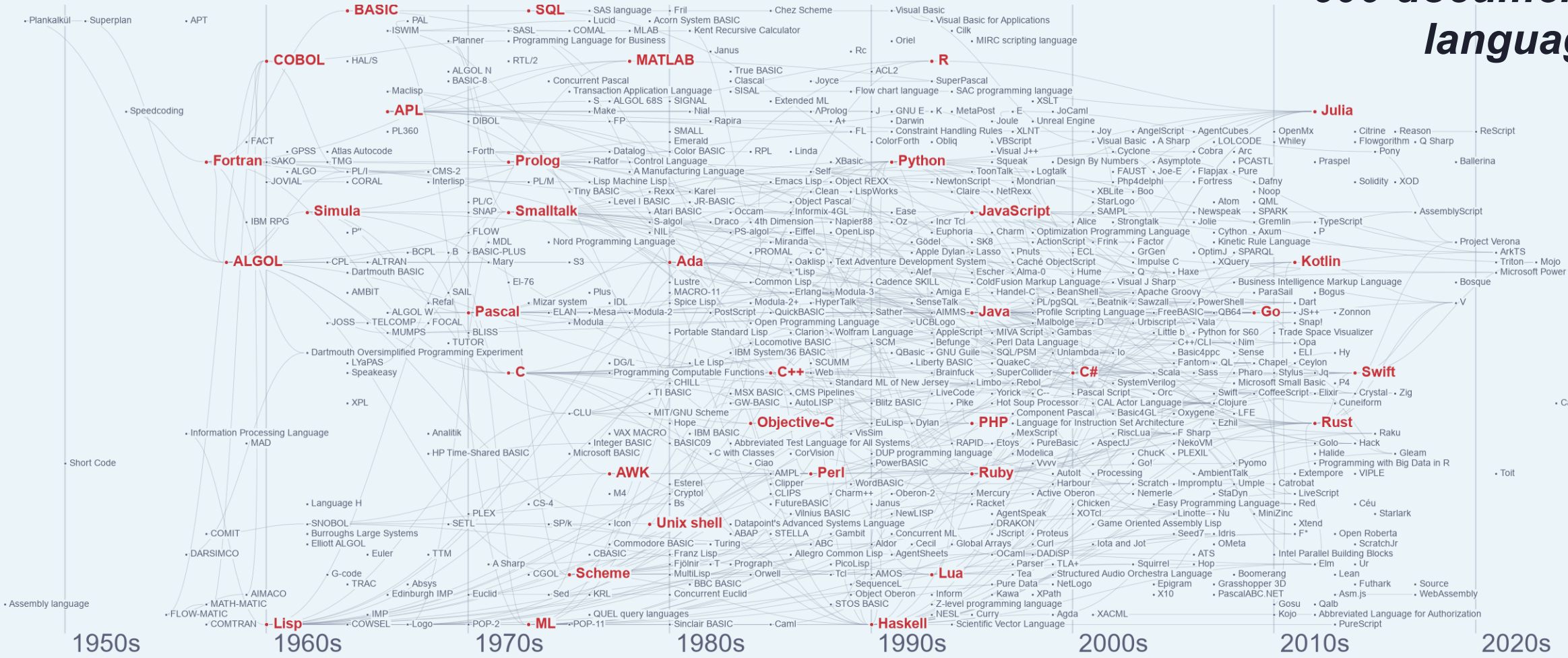
> Each language answered a pressure

Mainframes science, business, AI	**Time-sharing, Unix** many users, Unix tools, databases	**PCs and GUIs** large systems, defence	**The web** browsers and servers	**Multicore and phones** networked services, apps, safety



> Each language answered a pressure

>600 documented languages!



Languages are hard to kill: Fortran and COBOL are still used!

> Which languages are most visible online?

Nov 2026	Nov 2025	Change	Programming Language	Ratings	Change
1	1		Python	20.86%	-2.51%
2	2		C	11.48%	+1.80%
3	4	↑	Java	8.89%	+0.34%
4	3	↓	C++	8.35%	-0.60%
5	5		C#	3.39%	-4.26%
6	6		JavaScript	2.42%	-1.00%
7	12	⇈	R	2.27%	+0.60%
8	10	↑	SQL	1.91%	+0.11%
9	7	↓	Visual Basic	1.88%	-1.43%

TIOBE index: how visible each language is in web searches (engineers, courses, vendors).
Not the best language, nor the one with most code written. *tiobe.com/tiobe-index*

One task, four languages: Python

Mean temperature over the usable readings (`NA` = missing).
Predict: what output do you expect?

```python
# “data” is a list of lists containing weather.csv

total = 0.0
count = 0
for row in data:
    field = row[1]    # the second column
    if field != "NA":
        total += float(field)
        count += 1

print(f"mean {total / count:.2f} over {count} readings")
```

→ `mean 7.95 over 6 readings`

weather.csv

```
date,temperature_c
2025-11-01,8.1
2025-11-02,7.4
2025-11-03,NA
2025-11-04,9.9
2025-11-05,6.2
2025-11-06,10.3
2025-11-07,5.8
```

One task, four languages: C

The heart of the C version (**29 lines** in total, in the notebook):

```c
fgets(line, sizeof line, file);    /* discard the header */

while (fgets(line, sizeof line, file) != NULL) {
    strtok(line, ",");             /* the date, not needed */
    char *temperature = strtok(NULL, ",");
    if (temperature != NULL && strcmp(temperature, "NA") != 0) {
        total += atof(temperature);
        count += 1;
    }
}
fclose(file);
```

Declared first:

```c
char line[256];
double total = 0.0;
int count = 0;
```

Same four ideas: **a loop, a condition, a running total and a counter**.
New: C makes you say everything **out loud**.

One task, four languages: awk and SQL

awk: a language for tables of text

```
awk -F, 'NR>1 && $2!="NA" {total+=$2; n++} END {print total/n, n}' weather.csv
```

→ `7.95 6`

SQL: a language for asking questions of databases

```
query = ("SELECT AVG(temperature_c), COUNT(*) FROM readings"
         " WHERE temperature_c != 'NA'")
mean_temperature, usable = connection.execute(query).fetchone()
print(f"mean {mean_temperature:.2f} over {usable} readings")
```

→ `mean 7.95 over 6 readings`

Lines for the whole task: C 29, Python 11, awk 1, SQL 3.

Three ways to say the same thing

```
total, count = 0.0, 0
for row in rows:
    if row[1] != "NA":
        total += float(row[1])
        count += 1
print(round(total / count, 2))
```

Imperative: describe the steps

→ 7.95

```
usable = [float(row[1]) for row in rows
          if row[1] != "NA"]
print(round(sum(usable) / len(usable), 2))
```

Functional: build new values, never update a variable

→ 7.95

```
query = ("SELECT ROUND(AVG(temperature_c), 2)"
         " FROM readings"
         " WHERE temperature_c != 'NA'")
print(connection.execute(query).fetchone()[0])
```

Declarative: describe the result

→ 7.95

{ Try it yourself #1

Predict: what does this calculate? What is new compared with Python?

```
public static double mean(double[] values) {
    if (values.length == 0) {
        throw new IllegalArgumentException("no values");
    }
    double total = 0.0;
    for (int i = 0; i < values.length; i++) {
        total += values[i];
    }
    return total / values.length;
}
```

This is *Java*! The code calculates the **mean** of `values`, refusing an empty array.
New: types before names, braces, a three-part `for`, `public static`.
The rest, **you already know**.

About 2 minutes

Not every language is general purpose

Domain-specific languages: SQL, HTML and CSS, LaTeX, spreadsheet formulas, regular expressions.

```
import re

POSTCODE = re.compile(r"^[A-Z]{1,2}[0-9][0-9A-Z]? ?[0-9][A-Z]{2}$")

for text in ["EH8 9AB", "W1A 0AX", "eh8 9ab", "EH8 9A"]:
    print(repr(text), POSTCODE.match(text) is not None)
```

```
'EH8 9AB' True
'W1A 0AX' True
'eh8 9ab' False
'EH8 9A' False
```

The pattern is a small program in **another language**: Python's `re` runs it on its own engine (itself written in C).

Compact, at the cost of learning **another notation**.

Speed: where C wins

Sum the squares of 0 to 9,999,999, two ways.

```python
import numpy as np

def sum_of_squares(n):
    total = 0
    for i in range(n):
        total += i * i
    return total

values = np.arange(n, dtype=np.float64)
fast = (values * values).sum()
```

```
plain Python : 0.789 s
NumPy        : 0.0354 s
ratio        : 22 times faster
```

On one laptop: yours will differ.

The NumPy package hands the maths to compiled **C**: using NumPy is **calling C** from Python.

The price of speed

Python's integers grow as needed; NumPy's `int64` always uses 64 bits.

```
integers = np.arange(n, dtype=np.int64)
overflowed = int((integers * integers).sum())

print("Python's answer :", exact)
print("NumPy's int64   :", overflowed)
print("agree?          :", overflowed == exact)
print("largest int64   :", np.iinfo(np.int64).max)
```

```
Python's answer : 3333332833333335000000
NumPy's int64   : 12918900065630709 12
agree?          : False
largest int64   : 9223372036854775807
```

Watch out: no warning! Fixed-size integers are fast, but can **overflow** silently.

When do mistakes get caught?

C and Java are **compiled**: the whole program is checked **before it runs**. Python checks the syntax first, and everything else **when the line is reached**.

Python (dynamic)

```python
def risky(user_input):
    return "total: " + user_input

print(risky("ten"))
try:
    print(risky(10))
except TypeError as error:
    print("TypeError:", error)
```

Java (static)

```java
static String risky(String userInput) {
    return "total: " + userInput;
}

// risky(10);  rejected before running:
// int cannot be converted to String
```

Python finds it **while running**; Java **before** running.

```
total: ten
TypeError: can only concatenate str (not "int") to str
```

> How to choose a language

- Does the **platform** decide for me?
- Is there a **library** that already does most of the work?
- How **fast** must it be?
- How bad would a **bug** be?
- Is there **existing code**? Who will maintain it?
- Is the language still actively **developed**, with good documentation and tools?
- **Taste**: it matters too (though it should not be the main motivation)

Note: most of the time you won't choose. Finding your way around code you did not write is the valuable skill.

{ Try it yourself #2: what decides?

For each situation: **which question** from the last slide decides it?
Would you still **use Python**?
What could go wrong?

1. 40,000 lines of working Fortran simulate airflow. A new student wants to rewrite it all in Python
2. A supermarket turns each day's 500 million receipts into orders: the lorries must leave at 5 a.m.
3. Operating theatre: how much of a drug to give, and how fast, from the patient's weight
4. A chart for an assignment due tomorrow morning

{ Example solution #2

	Deciding question	Choice	What could go wrong?
1	Existing code?	Keep the Fortran, call it from Python	fewer and fewer people can fix it
2	Where does the time go?	Python, timed on a full day's receipts first	a busy day runs late: the lorries leave without orders
3	How bad would a bug be?	A language that checks more before it runs	checks catch some errors, not all: it still needs testing
4	Is there a library for this?	Python (e.g., using the matplotlib or seaborn package)	very little

> You learned programming, not just Python

- Breaking a problem into steps and functions
- Variables, types and expressions
- Repetition with loops
- Choosing a data structure that fits
- Reading an error message
- Naming, structuring and testing code

Python-specific: syntax, library names, tools (this is the easy part)

> “A language that doesn't affect the way you think about programming, is not worth knowing.”
> *Alan Perlis, Epigrams on Programming, 1982*

{ [Extra] Esoteric languages

In **Shakespeare**, programs are plays:

```
Hamlet:
 Thou art as lovely as the product of a large rural town and my amazing
 bottomless embroidered purse. Speak thy mind!
```

```
town = 2 * 2 * 1
purse = 2 * 2 * 2 * 1
print(town * purse, repr(chr(town * purse)))
```

Each noun is 1, each adjective doubles it.

→ `32 ' '` A whole sentence for one space!

```
(+/x)÷≢x
```

APL, not a joke: the mean of x in eight characters.

[Extra] A program you cannot see

```
···\t\n          # push 1
\n···\n          # label A:
·\n·             # copy it
\t\n·\t          # print number
···\t·\t·\n      # push 10 (newline)
\t\n··           # print character
···\t\n          # push 1
\t···            # add
·\n·             # copy it
···\t·\t\t\n     # push 11
\t··\t           # subtract
\n\t·\t\n        # if 0, go to B
\n·\n·\n         # go to A
\n··\t\n         # label B:
\n\n\n           # end
```

A complete program in **Whitespace**.

Only **spaces** (·), **tabs** (\t) and **newlines** (\n) count: every other character is ignored.

It counts from 1 to 10, just like:

```
for i in range(1, 11):
    print(i)
```

> Summary

Take-home messages:

- Short code, speed and safety pull against each other
- The same task can take one line in one language and thirty in another
- Languages answer outside pressures, such as hardware, cost and safety
- Speed can cost correctness: NumPy's overflow was silent
- Choose a language with a **reason**, and know **what could go wrong**
- You learned programming, not just Python: the ideas carry over

Homework: the consolidation challenge in the self-guided notebook.