Programming languages

objectives:

- to find the differences between few of the most powerful and popular languages in the world
- to find the problems and good qualities of each language
- make a language with the maximum number of good qualities of each
- to evaluate the newly created language and demonstrate how it solves the limitations found in existing languages
- to compare the structure, syntax, and use-cases of several major programming languages

languages

Programming languages are the tools we use to talk to computers. Every program starts as plain text written by a human. The computer itself can only understand electrical signals and binary, so the programming language acts like a translator. It turns our instructions into steps the machine can follow, such as calculating numbers, storing information, or reacting to user input. Different languages are built with different goals: some are designed to be easy to learn, some are built for speed, and others focus on flexibility or safety. By comparing these languages, we can understand how each one solves problems in its own way and why certain languages are better suited for certain tasks.

Compilers and interpreters

Computers only understand binary, so programming languages need a way to convert human-written code into something the machine can run. This is done through *compilers* and *interpreters*. A compiler reads the entire program and translates it into another form, such as bytecode (used by Java) or native code (used by C/C++). Bytecode is a middle layer that is not tied to any single computer, which is why Java programs can run on many systems. An interpreter, on the other hand, reads and executes the program step-by-step without creating a full compiled file. Interpreters are easier to test and change, while compilers run much faster once built. Many modern languages actually mix both ideas: for example, Java code is compiled into bytecode, and then the Java Virtual Machine interprets or further compiles it into the computer's native instructions this feature is used in our prototype language. Each method has strengths and weaknesses, and understanding them helps explain why different languages behave differently and where they are most useful.

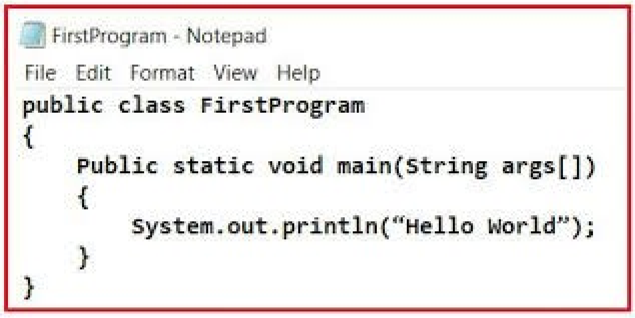
Note: construction of the prototype jappl has already begun and is in version 3.7
version 4 will come out shortly

Comparing languages

Java

Good qualities

- Very powerful and fast
- Used for Android apps, banking systems, large software
- Works on many devices (“write once, run anywhere”)



```
public class FirstProgram
{
    Public static void main(String args[])
    {
        System.out.println(“Hello World”);
    }
}
```

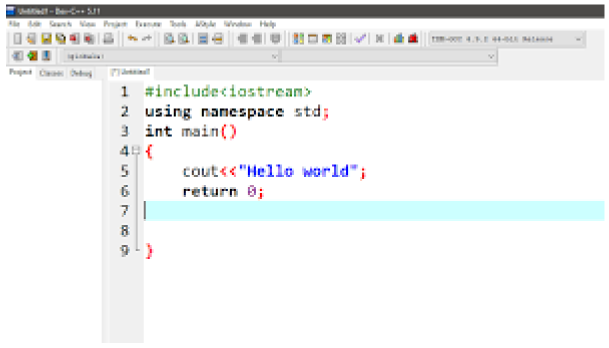
Bad qualities

- Harder for beginners
- Code is long and strict
- Requires setup and tools

C++

Good qualities

- Extremely Fast & Powerful
- Used in Real-World Systems
- Massive Library & Tool Support



```
#include<iostream>
using namespace std;
int main()
{
    cout<<"Hello world";
    return 0;

}
```

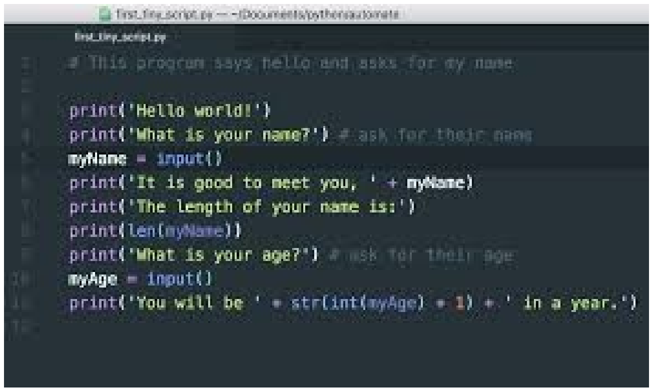
Bad qualities

- Very Hard for Beginners
- Manual Memory Management
- Easy to Introduce errors

Python

Good qualities

- Easiest real coding language
- Very readable, short code
- Used in AI, automation, data science, websites



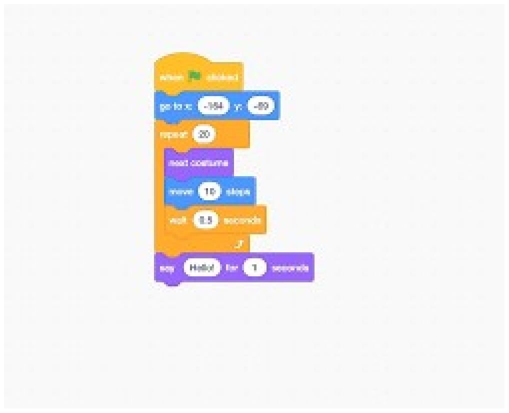
```
# This program says hello and asks for my name

print('Hello world!')
print('What is your name?') # ask for their name
myName = input()
print('It is good to meet you, ' + myName)
print('The length of your name is:')
print(len(myName))
print('What is your age?') # ask for their age
myAge = input()
print('You will be ' + str(int(myAge) + 1) + ' in a year.')
```

- Huge library support
- used in cybersecurity

Bad qualities

- Slower than Java
- Indentation can confuse beginners
- Not ideal for large mobile apps

Scratch



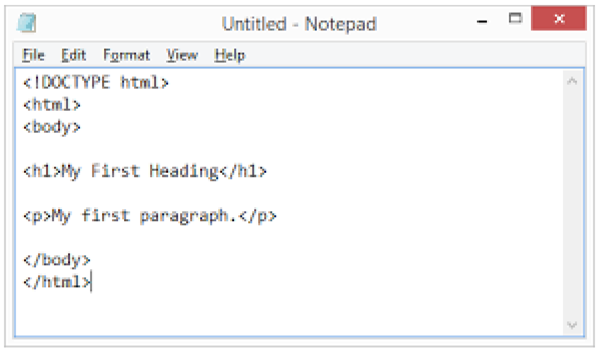
Good qualities

- Easiest to learn
- Block coding = no syntax errors
- Great for kids, games, animations

Bad qualities

- Limited — you can’t build real apps
- No real text coding
- Hard to scale big projects

HTML



```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>
</html>
```

Good qualities

- Simple
- Main language for structuring web pages
- Easy to create layouts, text, images, buttons

Bad qualities

- Not a real programming language
- No logic (no loops, variables, math, etc.) depends on javascript
- Needs CSS + JavaScript for real features

JAPPL (our own experimental language built in scratch)

(Text + text command based language)

Good qualities

- Beginner-friendly and flexible
- Syntax-tolerant (many ways to write the same command)
    - It automatically:
    - trims extra spaces
    - fixes missing quotes
    - identifies command tokens
    - converts text into a clean, normalized internal form
    - acts similarly to bytecode in Java/Python, but simplified.

Bad qualities

- Limited by Scratch's speed
- Not as powerful as Python or Java
- Still growing (fewer libraries, features)

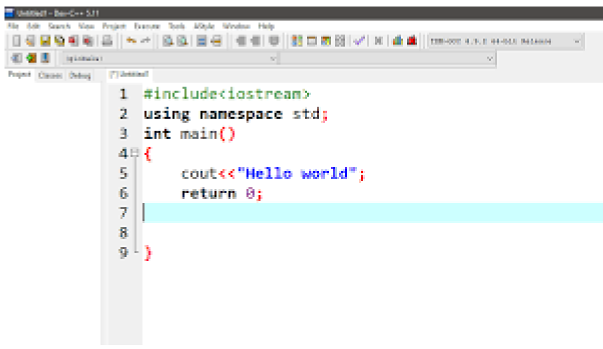
Quick Summary Table

Language	**Best At**	**Weakness**
Java	Big apps, performance	Hard for beginners
Python	Easy + powerful + real coding	Slower; strict indentation
Scratch	Learning basics	Not real apps
HTML	Websites	No logic
JAPPL	Middle step between Scratch + Python	Limited by Scratch

Creating our own language

the language must have the features of all above languages mainly:

- Easy to learn (scratch)
- Works on many devices ("write once, run anywhere") (java)
- Easiest real coding language and Very readable, short code (python)
- easy to access (html)
- light weight



```
#include<iostream>
using namespace std;
int main()
{
    cout<<"Hello world";
    return 0;


}
```

JAPPL Development Timeline

JAPPL 1

0.0 — Interpreter engine created

- JAPPL began as a simple text interpreter that read commands line-by-line.

0.7 — Text engine added

- Introduced basic text parsing so the program could read user-typed commands.

0.9 — Converted into a compiler

- Instead of running commands directly, JAPPL now compiles them into an internal list (similar to bytecode).

1.3.3 — Core commands implemented

- Added essential functions:
  • `wait`
  • screen text coordinates
  • `clear screen`
- First stable version.
- **Shared publicly**.

JAPPL 2 — Commands & Usability

2.5 — Variable system introduced

- Added:
  • `make variable`
  • `set variable`
  • `ask` (input)

2.6 — Operator engine introduced

- Early math/logic operators added.
- Internal calculator logic partially reused from NTPrograms (credited).

2.9 — GUI improvements and fixes

- Multiple interface and stability updates.

JAPPL 3 — True Language Features

3.0.0 — Operator compatibility overhaul

- All commands now support operators and variable expressions.
- Improved how variables interact with numbers and strings.

3.1.0 — Multiple-variable expressions

- Operators can now handle more than one variable per expression.
- Variables and numbers accepted in any order.

3.4.0 — Debugging improvements

- Added compiler logs for tracking errors and behaviour.
- Added `delete variable` feature.

3.6.0 — Sensing + Loops added

- New sensing variables:
  • mouse x, mouse y
  • date/time (hour, day, month, year, etc.)
- All sensing values stored as JAPPL variables.
- Added `repeat` and `forever` loop

3.6.0 – 4.0.0

- fixed operators
- increased overall speed of execution

JAPPL is a lightweight, beginner-friendly **text-based programming language** designed to teach real coding concepts without strict syntax rules. It uses a **flexible parser**, meaning commands like `print hi`, `print=hi`, or `print"hi"` are all accepted. it has a **compiler**, **instruction list**, **virtual machine**, **variable/RAM system**, **operator engine**, **sensing engine**, and **error logging**.

Its goal is to be a “bridge language” between very simple environments and real programming languages — letting beginners learn text commands, logic, variables, and expressions in a forgiving environment before moving on to Python, Java, or C++.

How JAPPL Is Different From Other Languages

JAPPL vs Python

Python:

- Easy, clean syntax
- Powerful libraries (AI, web, automation)
- Automatically manages memory

Difference:
JAPPL is *even more forgiving* than Python — syntax doesn’t need to be perfect, and there are fewer rules.

Python is a real general-purpose language, while JAPPL is a small training language designed to teach fundamentals gently.

🔵 JAPPL vs Java

Java:

- Fast, strongly typed, strict
- Used for Android, banking systems, large applications
- Very structured

Difference:
JAPPL is the opposite of strict. It avoids complex rules, doesn’t require types, and focuses on simplicity. Java requires full structure (classes, methods), while JAPPL runs simple line-by-line commands.

JAPPL vs C++

C++:

- Extremely fast and powerful
- Manual memory management
- Very hard for beginners
- Used for engines, OS development, hardware-level work

Difference:
C++ gives total control but is very complex. JAPPL gives *no* memory responsibility to the user and is designed to prevent crashes or dangerous behavior. JAPPL is educational; C++ is industrial and low-level.

JAPPL vs Lua

Lua:

- Lightweight scripting language
- Used in games (Roblox, engines)
- Fast, simple, flexible

Difference:
Lua is a real scripting language used in professional tools. JAPPL behaves similarly in structure (parsing → bytecode → VM), but is much smaller and built for learning, not production. Lua has loops, functions, tables; JAPPL currently focuses on straightforward commands.

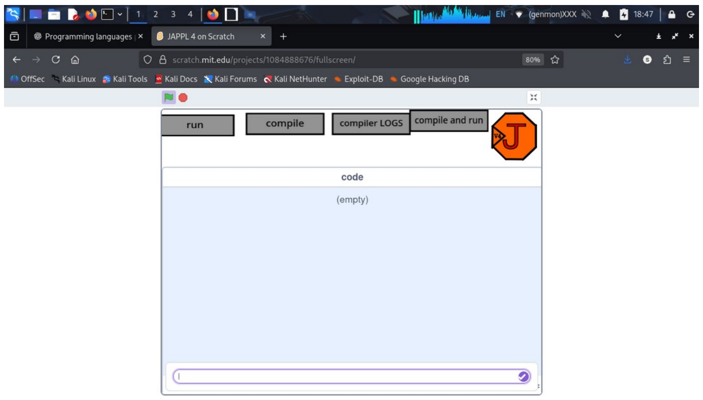
Jappl as an engine

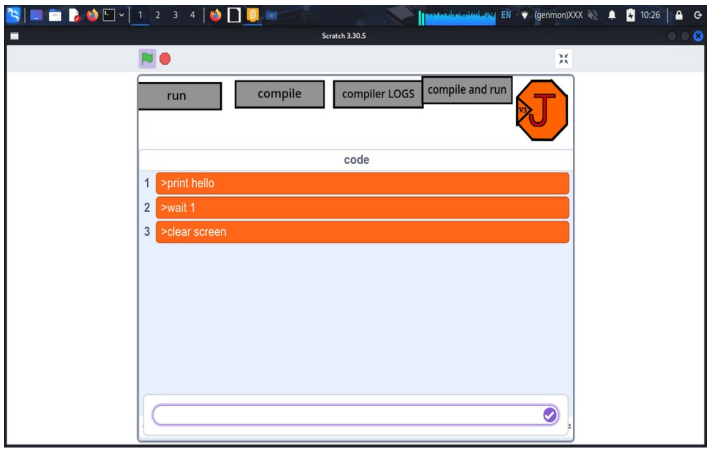
JAPPL is a standalone scripting language with its own VM and editor, designed to become a full engine ecosystem.

Most languages (java c++) can only run one script at a time jappl is planned to use executer cloners to clone the executer and compiler to run multiple scripts

example code

```
>ask first number?

>make variable n1

>set variable n1 to [ask]

>ask second number

>make variable n2

>set variable n2 to [ask]

>ask 3rd number

>print ([n1]+[n2]+[ask])
```





```
>print hello
>wait 1
>clear screen
```

Official JAPPL documentation (Scratch forums):
https://scratch.mit.edu/discuss/topic/837372/
JAPPL experimental project (Scratch project page):
https://scratch.mit.edu/projects/1235397785/