1) Python

What is Python?

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

2) C++

is a general-purpose programming language created by Bjarne Stroustrup as an extension of the C programming language, or "C with Classes". The language has expanded significantly over time, and modern C++ now has object-oriented, generic, and functional features in addition to facilities for low-level memory manipulation. It is almost always implemented as a compiled language, and many vendors provide C++ compilers, including the Free Software Foundation, LLVM, Microsoft, Intel, Oracle, and IBM, so it is available on many platforms.[9]

C++ was designed with an orientation toward system programming and embedded, resource-constrained software and large systems, with performance, efficiency, and flexibility of use as its design highlights.[10] C++ has also been found useful in many other contexts, with key strengths being software infrastructure and resource-constrained applications,[10] including desktop applications, video games, servers (e.g. e-commerce, web search, or databases), and performance-critical applications (e.g. telephone switches or space probes).[11]

C++ is standardized by the International Organization for Standardization (ISO), with the latest standard version ratified and published by ISO in December 2020 as *ISO/IEC 14882:2020* (informally known as C++20).[12] The C++ programming language was initially standardized in 1998 as *ISO/IEC 14882:1998*, which was then amended by the C++03, C++11, C++14, and C++17 standards. The current C++20 standard supersedes these with new features and an enlarged standard library. Before the initial standardization in 1998, C++ was developed by Danish computer scientist Bjarne Stroustrup at Bell Labs since 1979 as an extension of the C language; he wanted an efficient and flexible language similar to C that also provided high-level features for program organization.[13] Since 2012, C++ has been on a three-year release schedule[14] with C++23 as the next planned standard.[15]

3) DATABASE

What’s a database?

A database can be thought of as a kind of electronic filing cabinet; it contains digitized information (“data”), which is kept in persistent storage of some kind, typically on magnetic disks. Users can insert new information into the database, and delete, change, or retrieve existing information in the database, by issuing *requests* or *commands* to the software that manages the database—which is to say, the database management system (DBMS for short). *Note:* Throughout this book, I take the term *user* to mean either an application programmer or an interactive user[2] or both, as the context demands.

Now, in practice, those user requests to the DBMS can be formulated in a variety of different ways (e.g., by pointing and clicking with a mouse). For our purposes, however, it’s more convenient to assume they’re expressed in the form of simple text strings in some formal language. Given a human resources database, for example, we might write