Advanced Data Programming

Introduction

Karl Ho
School of Economic, Political and Policy Sciences
University of Texas at Dallas

Overview:

- What is programming?
- What is data programming?
- History of Computing
- Social Science and data programming
- Workshop: Introduction to Git and GitHub

What is programming?

- Programming is a practice of using programming language to design, perform and evaluate tasks using a computer. These tasks include:
  - Computation
  - Data collection
  - Data management
  - Data visualization
  - Data modeling

} Data programming

- In this course, we focus on data programming, which emphasizes programs dealing with and evolving with data.

Why learning programming Languages?

- Understand the differences between apparently similar constructs in different languages
- Be able to choose a suitable programming language for each application
- Enhance fluency in existing languages and ability to learn new languages
- Application development

- Maribel Fernandez 2014

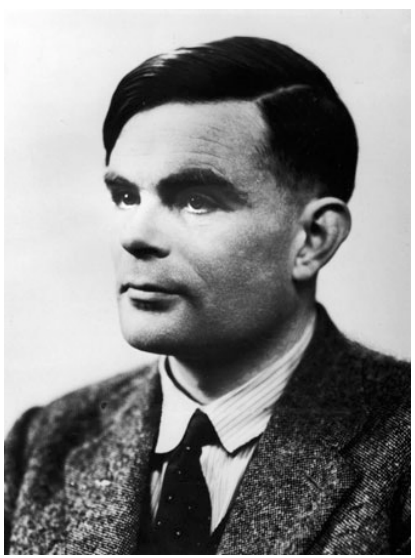
History of Computing

Source: Britannica

- Niels Henrik Abel 1800s
- Sophus Lie (pronounced Lee)
- Charles Babbage (1791–1871)
- Alan Turing (1912–1954)
  - "On Computable Numbers, with an Application to the Entscheidungsproblem"
  - Turing machine
  - Halting problem
- John Atanasoff, Iowa State University invented in 1939
  - First computer, the ABC or Atanasoff-Berry Computer
  - Can perform logical and other mathematical operations but lack a central processing unit

History of Computing

- Electronic Numerical Integrator and Computer (ENIAC) was unveiled in 1946 by John Mauchly and J. Presper Eckert.
  - ENIAC was difficult to program since the program was written by plugging cables into a switch, similar to an old telephone switchboard.
- Electronic Discrete Variable Automatic Computer (EDVAC) was introduced and John von Neumann proposed storing the computer programs on EDVAC in memory along with the program data.
- von Neumann machine: Stored program computer
  - In the stored-program model, the program and data are stored in memory.
  - The program manipulates data based on some input. It then produces output.



Source: Britannica

Von Neumann machine

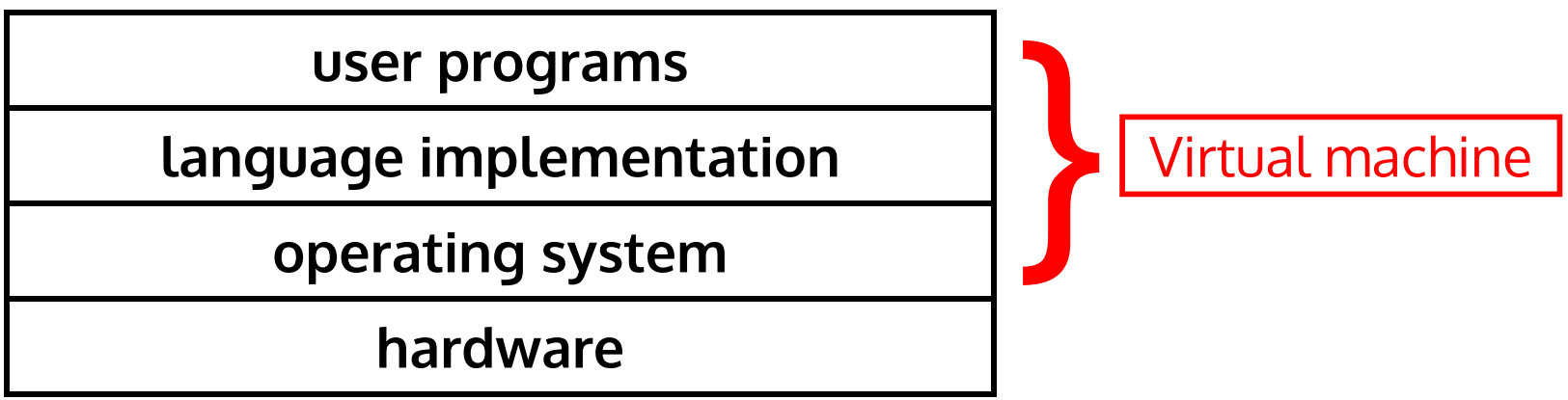
1. Memory, which contains data and machine instructions represented as sequences of bits;
2. Processor (also called Central Processing Unit, CPU), which consists of an arith- metic logic unit with a set of processor registers, and a control unit with an instruction register and program counter;
3. Peripherals, which may include a keyboard, display, mouse, printer, etc;
4. File systems, which permit to store data externally.

More terms

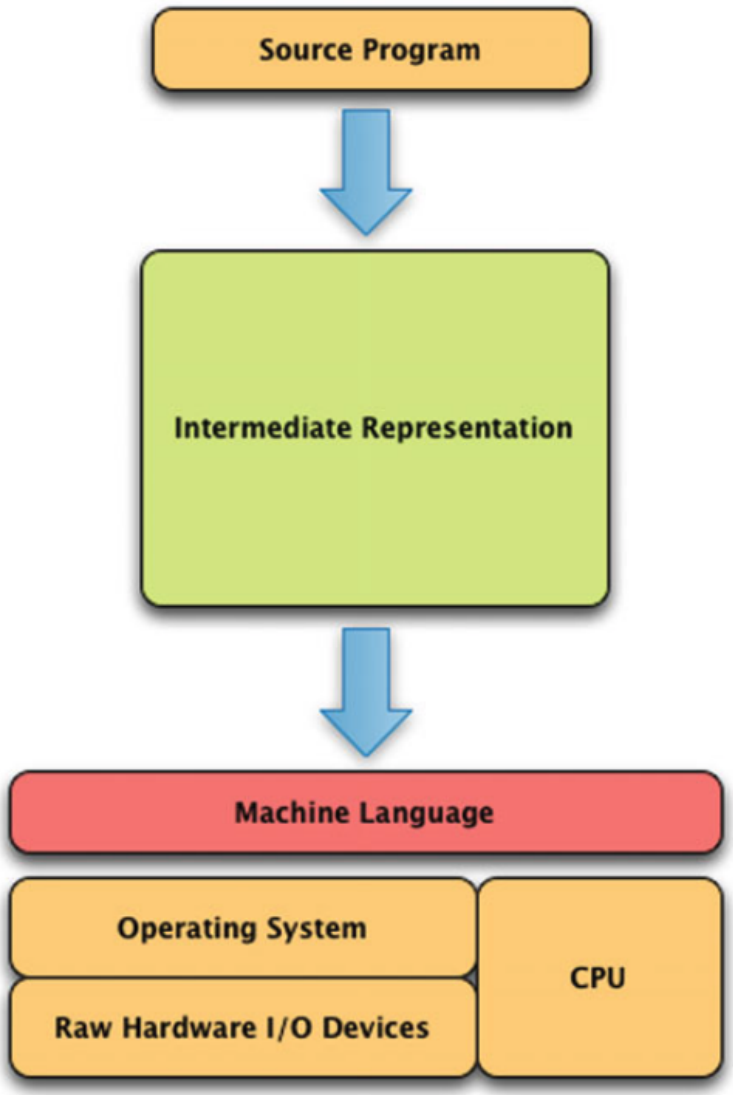
- Machine language: the set of machine instructions for arithmetic and logic operations used by the processor
- Operating system: the system supplies higher-level primitives than those of the machine language (e.g. input/output operations, file management system, editors).
  - For example, Linux, Unix, MS DOS and Mac OS are operating systems.

Language implementations

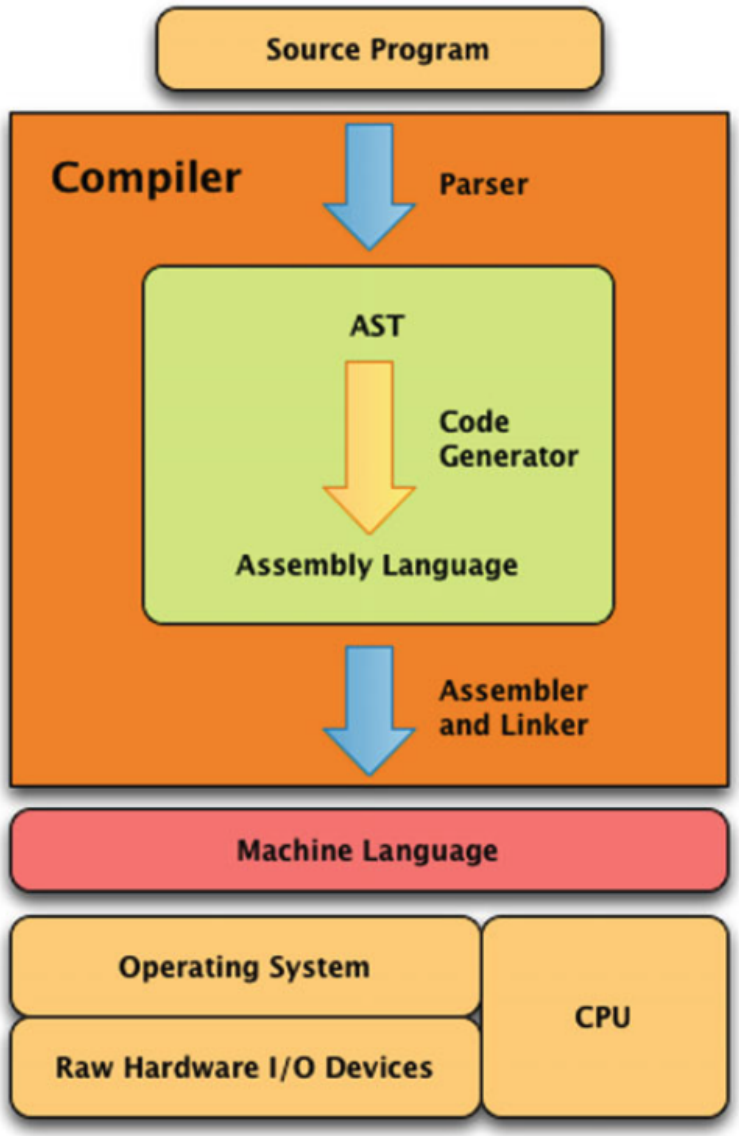
- *Language implementations* are built on top of these, and user programs or applications form the top layer, as shown in the following:



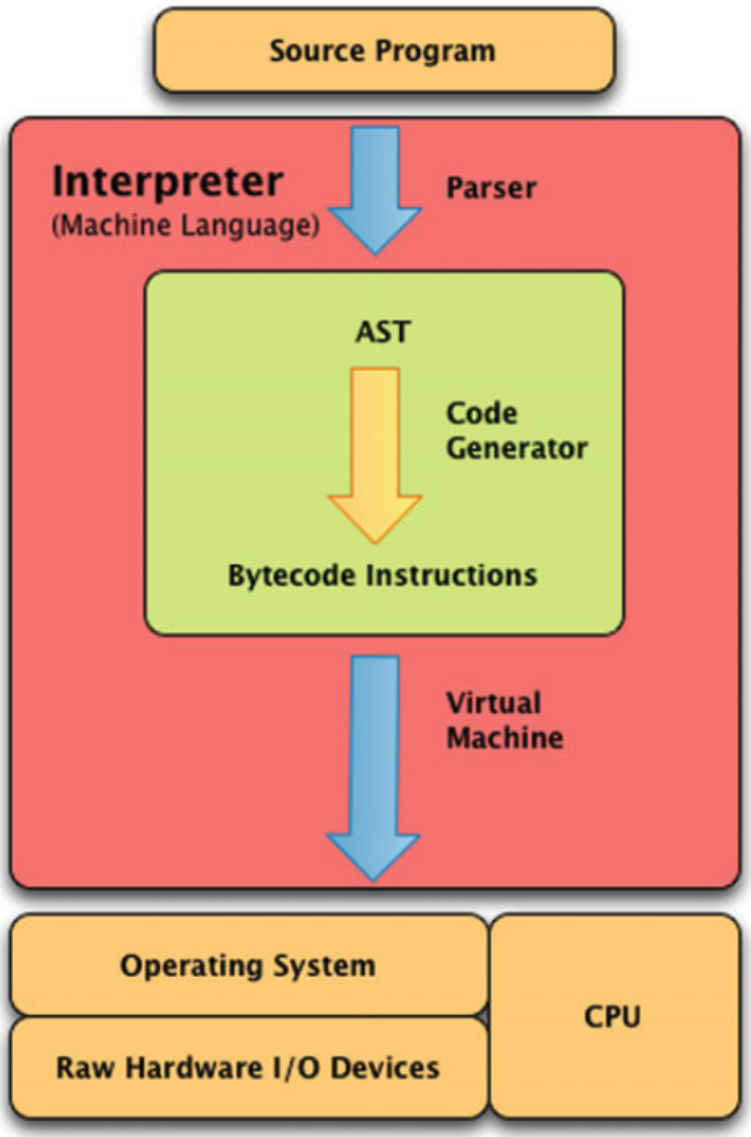
Language implementations



Compilation



Interpretation



Low-level languages

- Machine language
  - Assembly language
  - C

Systems languages

- BASIC
  - REALbasic
  - Visual Basic
- C++
- Objective-C
  - Mac
- C#
  - Windows
- Java

Scripting languages

- Perl
- Tcl
- JavaScript
- Python

Programming Languages

- High level or low level—Relates to abstraction, i.e., how close your instructions correspond to what computers actually do.
- Object-oriented—Relates to the design of programs consisting of objects, their properties, and their functionality, along with their interactions with each other.
- Static or dynamic—Relates to the strength of the association of the type to the objects defined.
- Functional programming—Relates to the level of significance that functions hold in a language, i.e., are they treated the same way as other objects or not (e.g., assigning to variables, passing as arguments to functions, immutability of variables, etc.)

Hardware architectures

- Register-based central processing units
- Stack-based virtual machines

Programming paradigms

Programming paradigms are ways of thinking about programming

1. Object-oriented/imperative programming
2. Functional programming
3. Logic programming.

Imperative languages

- Programs are decomposed into computation steps (also called commands, statements, or instructions), reflecting the step-wise execution of programs in usual hardware.
- Subprograms (also called routines, procedures, etc.) are used to build programs in a modular way.
- Programs written in imperative languages give accurate descriptions of how to solve a given problem.
- For example: Fortran, Algol, Pascal, C, Python and Java

Functional languages

- Programs are functions, which can be composed to build new functions as in the mathematical theory of functions (the basis of these languages).
- Also called declarative, since the focus is on what should be computed, not how it should be computed.
- Functional languages emphasize the use of expressions, which are evaluated by simplification.
- LISP, introduced by John McCarthy in the 1950s, is considered to be the ancestor of all functional programming languages.
- Other examples: Haskell, SML, Caml and Clean

Object-oriented languages

- Programs are collections of objects which can only be accessed through the operations (or methods) provided for them, and are usually hierarchically organized.
- An object can be thought of as an *entity* combining data (fields) and operations (methods).
- In the object-oriented design methodology, the designer produces a hierarchical description of the structure of the system, which is the basis for the implementation in an object-oriented language.
- Object-orientation is sometimes considered as a feature of imperative languages, however, it can also be found in functional languages and can be combined with logic languages.
- For example: Java and Python

Logic languages

- Programs describe a problem rather than defining an algorithmic implementation.
- This class of languages is declarative, since the focus is on the specification of the problem that needs to be solved and not on how it is solved.
- A logic program is a description (usually using the language of first-order logic) of facts and properties about a problem.
- The most well-known logic programming language is Prolog, which was designed by Colmerauer, Roussel and Kowalski in the 1970s.
- More modern logic programming languages combine logic programming and constraint-solving.

Components of a Programming Language

A programming language has three main components:

1. *Syntax*
2. *Semantics*
3. *Implementation*

Components of a Programming Language

1. Syntax

- defines the form of programs.
- It describes the way expressions, commands and declarations are built and put together to form a program.

Components of a Programming Language

2. Semantics

- gives the meaning of programs.
- It describes the way programs behave when they are executed.

Components of a Programming Language

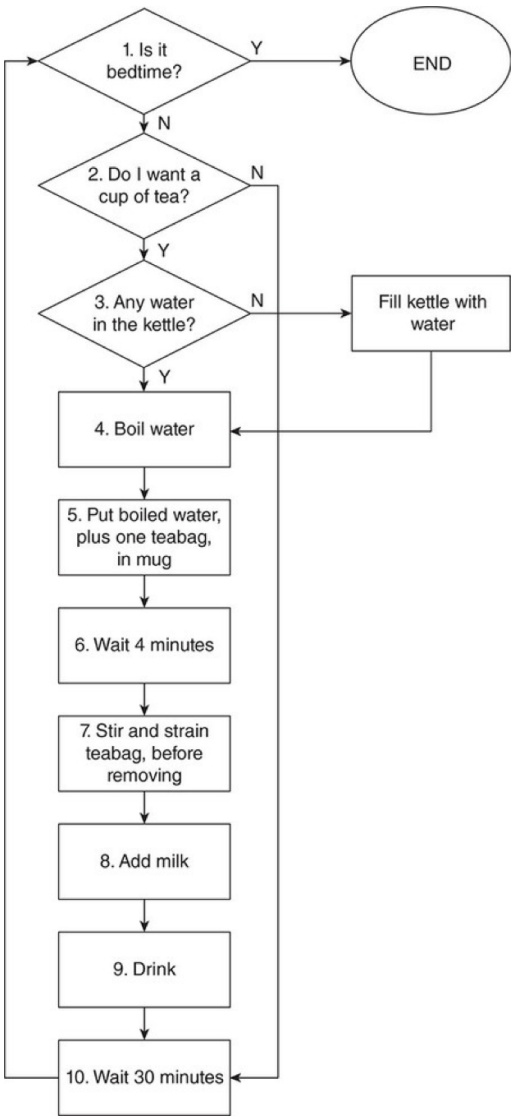
3. Implementation

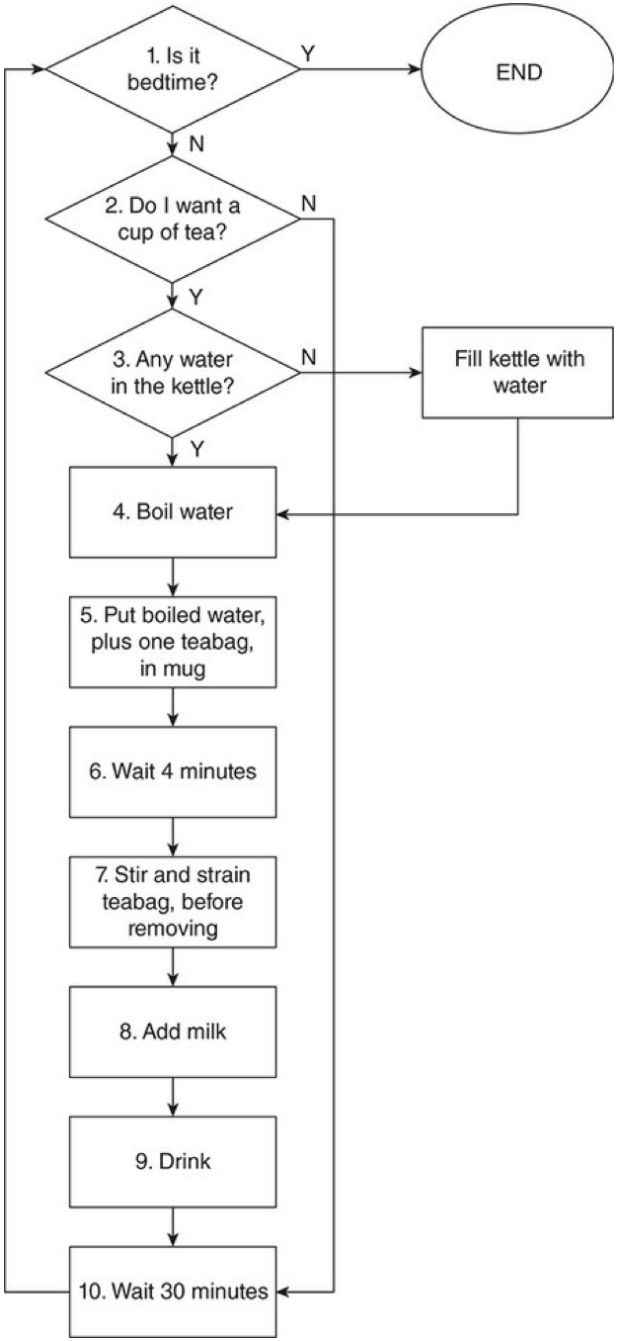
- a software system that can read a program and execute it in a computer.
- It usually consists of a compiler or interpreter, plus a set of tools, such as editors and debuggers, which help programmers write and test programs.
- Programming languages usually include a runtime support, which provides functionalities such as memory management primitives (e.g., a garbage collector), mechanisms to implement calls to subprograms including parameter passing, etc.

Algorithm

1. Put water in kettle
2. Boil water in kettle
3. Put boiled water, plus one teabag, in mug
4. Wait 4 minutes
5. Stir and strain teabag, before removing
6. Add milk.

Algorithm


1. Is it bedtime?
Y
END
N
2. Do I want a cup of tea?
N
Y
3. Any water in the kettle?
N
Fill kettle with water
Y
4. Boil water
5. Put boiled water, plus one teabag, in mug
6. Wait 4 minutes
7. Stir and strain teabag, before removing
8. Add milk
9. Drink
10. Wait 30 minutes


1. Is it bedtime?
Y
END
N
2. Do I want a cup of tea?
N
Y
3. Any water in the kettle?
N
Fill kettle with water
Y
4. Boil water
5. Put boiled water, plus one teabag, in mug
6. Wait 4 minutes
7. Stir and strain teabag, before removing
8. Add milk
9. Drink
10. Wait 30 minutes

*Code and Data for the Social Sciences: A Practitioner's Guide (*Gentzkow and Shapiro 2014)

1. Automation
    - For replicability (the future you)
2. Version Control
    - Allow evolution and updated editions
3. Directories
    - Organize by functions
4. Keys
    - Index variable (relational)
5. Abstraction
    - KISS
6. Documentation
    - Comments for communicating to later users
7. Management
    - Collaboration ready

How do professional software developers code?

Agile software development describes an approach to software development under which requirements and solutions evolve through the collaborative effort of self-organizing cross-functional teams and their customer(s)/end users(s). It advocates adaptive planning, evolutionary development, early delivery, and continuous improvement, and it encourages rapid and flexible response to change.

Source: Wikipedia on Agile Software Development

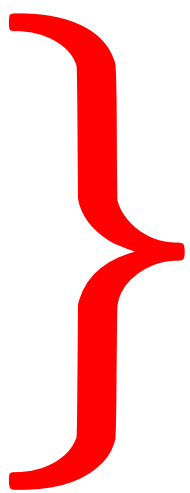
DRY – Don't Repeat Yourself

Write a function!

Function example

```
# Create preload function
# Check if a package is installed.
# If yes, load the library
# If no, install package and load the library

preload<-function(x)
{
  x <- as.character(x)
  if (!require(x,character.only=TRUE))
  {
    install.packages(pkgs=x,
repos="http://cran.r-project.org")
    require(x,character.only=TRUE)
  }
}
```



For replicability and collaboration

Use Version Control System!

Agile programming

- Individuals and interactions
  - It is better to have a good team of developers who communicate and collaborate well, rather than a team of experts each operating in isolation. Communication is a fundamental concept.
- Working software
  - comment inline with the code and to keep external documentation light
- Customer collaboration
  - progressively elaborated and adapted based on feedback
- Responding to change
  - quick responses to change and continuous development

Source: Wikipedia on Agile Software Development

Why learning programming?

- For social scientists, programming is more than just developing an application. It could introduce social injustice:
  - sexist
  - malicious
  - offensive
  - racist
  - discriminative

Question:

Why are the history of Mathematics and Computer Science so closely tied together?

Answer:

The problems in Mathematics were growing complex enough that many mathematicians were developing models and languages for expressing their algorithms. This was one of the driving factors in the development of computers and Computer Science as a discipline.

" focusing on the digital world offers social science the potential not only to extend its reach into new forms of sociality, but also to deepen our understanding of existing forms:

" "these technologies and their allied data have the potential to 'digitally-remaster' classic questions about social organization, social change and the derivation of identity from collective life"

- Housley *et al.* 2014

" learning how to program can significantly enhance how social scientists can think about their studies, and especially those premised on the collection and analysis of digital data.

- Brooker 2019:

" The complexity for minimum component costs has increased at a rate of roughly a factor of two per year...Certainly over the short term this rate can be expected to continue, if not to increase.

- Gordon Moore, 1965

" ...Then you better start swimmin'...Or you'll sink like a stone...For the times they are a-changin'.

- Bob Dylan

" Chances are the language you learn today will quite likely not be the language you'll be using tomorrow.

Python

- functional
- compiled
- interpreted
- object-oriented (class-based)
- imperative
- metaprogramming
- extension
- impure
- interactive mode
- iterative
- reflective
- scripting

R

- array
- interpreted
- impure
- interactive mode
- list-based
- object-oriented (prototype-based)
- scripting

1. Mutable data entity
2. Immutable data entity
3. Independent explicit pointers/reference
4. Extra precision basic types
5. Strings
6. Arrays/indexed sequences
7. Hash table/maps/key-value table
8. Named tuples/record/struct
9. Ordinal type
10. Sets
11. Recursive data types/lists
12. Universal polymorphism/subtype
13. Objects
14. Class and inheritance (single or multiple inheritance)
15. Modules/package/namespace

Languages	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ALICE	▪	X	X	X	▪	▪	X	X	X	X	▪	X	M	▪	X
Ada 2005	▪	Δ	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪
C	▪	X	▪	▪	▪	▪	X	▪	▪	S	▪	X	X	X	▪
C++	▪	X	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪
C#	▪	X	▪	Δ	▪	▪	▪	▪	▪	S	▪	▪	▪	▪	W
Chapel	▪	X	X	▪	▪	▪	▪	▪	▪	▪	X	▪	▪	▪	▪
Claire	▪	▪	X	X	▪	▪	X	▪	ρ	▪	▪	▪	▪	▪	▪
Clojure	L	▪	R	L	▪	▪	▪	▪	ρ	▪	▪	▪	▪	▪	▪
ECLiPSe	X	▪	▪	D	▪	▪	▪	▪	S	▪	▪	▪	▪	▪	▪
Emerald	▪	▪	X	X	▪	▪	X	▪	▪	▪	▪	▪	▪	X	▪
Estrel	▪	X	X	X	X	X	X	X	X	X	X	X	X	X	▪
F#	▪	▪	R	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪
Fortran 2008	▪	X	▪	▪	▪	▪	X	X	▪	X	▪	▪	▪	▪	▪
Haskell	X	▪	X	▪	▪	▪	S	▪	▪	▪	▪	▪	▪	▪	▪
Java	▪	X	R	▪	▪	▪	▪	X	▪	▪	S	▪	▪	▪	▪
Javascript	▪	Δ	X	D	▪	▪	▪	X	S	X	X	X	▪	X	X
Lisp	▪	▪	X	D	▪	▪	▪	S	V	S	▪	▪	V	V	X
Lua	▪	▪	X	X	▪	▪	▪	X	S	S	▪	V	▪	S	▪
ML	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	X	X	▪
Modula-3	▪	X	▪	▪	▪	▪		▪	▪	▪	▪	▪	▪	▪	▪
Perl	▪	X	L	L	▪	▪	▪	S	▪	S	S	▪	▪	▪	▪
PhP	▪	X	X	X	▪	▪	▪	X	X	X	X	L	▪	▪	▪
Python	▪	▪	A	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪
Ruby	▪	▪	R	D	▪	▪	▪	X	ρ	▪	ρ	▪	▪	▪	▪
Scala	▪	▪	X	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪	▪
SMIL	▪	X	X	X	▪	X	X	X	X	X	X	X	M	X	▪
X10	▪	▪	X	▪	▪	▪	X	▪	X	L	▪	▪	▪	▪	▪

■—built-in/built-in library;
A—aliases used as references;
Δ—limited;
D—dynamically typed;
L—through library;
M—media objects;
R—reference instead of pointer;
ρ—range,
S—simulated by other data structure;
V—variations of the language support;
X—absent

Source: Bansal, Arvind Kumar. 2013. *Introduction to programming languages*. CRC Press.

" Beware of bugs in the above code; I have only proved it correct, not tried it."

- Donald Knuth, author of *The Art of Computer Programming*