Introduction to Programming Languages

Ryan Stansifer

Computer Sciences
Florida Institute of Technology
Melbourne, Florida USA  32901

http://www.cs.fit.edu/~ryan/

14 October 2022
Science is what we understand well enough to explain to a computer; art is everything else.

Knuth, “Things a Computer Scientist Rarely Talks About”
The intellectual roots of computer science are not merely bookkeeping, astronomy, and ballistics. Computer Science is the child of logic, the mathematical sciences, and human imagination.

Overview of Introduction

• Abstraction: language versus linguistics, abstraction, complexity, visualization versus expression, notation, Sapir-Whorf
• Paradigms and Aspects
• Ancient History of Written Expression
• Development of Mathematical Language
• History of Programming Languages: (I) FORTRAN; (II) COBOL, SNOBOL, etc; (III) PL/I, Pacal, etc; (IV) Ada, Java, etc
Abstraction

Let’s think about what *abstraction* means. When we learn about the world around us, we all make mistakes. For example, we call palm trees by that name because they are tall plants, even though botanists tell us that they are related to grasses and not to trees. For another example, consider when we learn English we learn the pattern of the past tense of verbs:

- play    played
- work    worked
- move    moved
- open    opened

But what is the past tense of “to go,” or “to run?”
Abstraction

Other examples of analogy in English linguistics . . .
What is the plural of the obscure words ziff, zo, and zax? Why, ziffs, zos, and zaxes, of course. The required plurals are formed according to a pattern already familiar from a large number of other English nouns. Consider the plural of a small class of nouns derived from Latin:

<table>
<thead>
<tr>
<th>Singular</th>
<th>Plural</th>
</tr>
</thead>
<tbody>
<tr>
<td>cactus</td>
<td>cacti</td>
</tr>
<tr>
<td>hippopotamus</td>
<td>hippopotami</td>
</tr>
<tr>
<td>radius</td>
<td>radii</td>
</tr>
<tr>
<td>succubus</td>
<td>succubi</td>
</tr>
<tr>
<td>syllabus</td>
<td>syllabi</td>
</tr>
</tbody>
</table>

What is the plural of octopus?
AN ACCIDENTAL BITE BY A RADIOACTIVE SPIDER HAS GIVEN TEENAGER PETER PARKER SUPER POWERS, AND TRANSFORMED HIM INTO--

the AMAZING SPIDER-MAN

SPIDERS ARE SMALL AND FAST, DOC...

OCTOPUSES ARE BIG AND CLUMSY.

YOU'RE WRONG ABOUT THAT!

I KNOW, I SHOULD HAVE SAID "OCTOPUS!!"
Many people would say *octopi*, but because the word is of Greek origin, its Greek plural is *octopodes*. The problem is three competing plural patterns: English, Latin, Greek. Fowler’s Modern English Usage states that “the only acceptable plural is English is *octopuses,*” and that *octopi* is misconceived and *octopodes* is pedantic. *Octopi* derives from the mistaken notion that *octōpūs* is a second declension Latin noun, which it is not. It is originally from the ancient Greek masculine noun *oktopous* (οκτωπούς), whose plural is *oktopodes* (οκτωποδες). Examples from Trask, *Historical Linguistics*, page 106.

In this class we are *pedantic*, or we prefer to say *careful* and *precise*. We are precise for two reasons: for the sake of scientific accuracy, and because computers are stupid and unforgiving.
How many have studied a foreign language?
<table>
<thead>
<tr>
<th><strong>English</strong></th>
<th><strong>German</strong></th>
<th><strong>Spanish</strong></th>
<th><strong>Turkish</strong></th>
<th><strong>Hungarian</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>Datei</td>
<td>Fichero</td>
<td>Dosya</td>
<td>Állomány</td>
</tr>
<tr>
<td>New</td>
<td>Neu</td>
<td>Nuevo</td>
<td>Yeni</td>
<td>Új</td>
</tr>
<tr>
<td>Open</td>
<td>Öffnen</td>
<td>Abrir</td>
<td>Aç</td>
<td>Megnyitás</td>
</tr>
<tr>
<td>Close</td>
<td>Schließen</td>
<td>Cerrar</td>
<td>Kapat</td>
<td>Bezárás</td>
</tr>
<tr>
<td>Save</td>
<td>Speichern</td>
<td>Salvar</td>
<td>Kaydet</td>
<td>Mentés</td>
</tr>
<tr>
<td>Print</td>
<td>Drucken</td>
<td>Imprimir</td>
<td>Yazdir</td>
<td>Nyomtatás</td>
</tr>
<tr>
<td>Exit</td>
<td>Beenden</td>
<td>Salir</td>
<td>Çikiş</td>
<td>Kilépés</td>
</tr>
<tr>
<td>Edit</td>
<td>Bearbeiten</td>
<td>Editar</td>
<td>Değiştir</td>
<td>Szerkesztés</td>
</tr>
<tr>
<td>Undo</td>
<td>Rückgängig</td>
<td>Anular</td>
<td>Geri Al</td>
<td>Vissza</td>
</tr>
<tr>
<td>Redo</td>
<td>Wiederholen</td>
<td>Rehacer</td>
<td>Tekrarla</td>
<td>Újra</td>
</tr>
<tr>
<td>Cut</td>
<td>Ausschneiden</td>
<td>Cortar</td>
<td>Kes</td>
<td>Kivág</td>
</tr>
<tr>
<td>Copy</td>
<td>Kopieren</td>
<td>Copiar</td>
<td>Kopyala</td>
<td>Másol</td>
</tr>
<tr>
<td>Paste</td>
<td>Einfügen</td>
<td>Pegar</td>
<td>Yapistir</td>
<td>Beilleszt</td>
</tr>
</tbody>
</table>
How many have studied a linguistics?
What is linguistics?

In linguistics, language is the subject, not a particular language. Likewise, in the study of programming languages, it is the medium (programming language) that is the subject, not how to program in some language. To quote an aphorism by Marshall McLuhan (1911–1980), anointed “patron saint” by Wired magazine: “the Medium Is the Message.” Usually it is important to write programs to accomplish some task. Often in this class, we are not interested program’s task, but rather the medium in which it is written.
Linguistics: the study of the units, nature, structure, and modification of language. Not a language, the difference between the study of a language and linguistics is abstraction.
Linguistics: the study of the units, nature, structure, and modification of language. Not a language, the difference between the study of a language and linguistics is *abstraction*.

Concepts in linguistics: grammar, word order (SVO, etc.), i.a.

**Case:**
Linguistics: the study of the units, nature, structure, and modification of language. Not a language, the difference between the study of a language and linguistics is abstraction.

Concepts in linguistics: grammar, word order (SVO, etc.), i.a.

Case: indicates the main role of a construct in a sentence, e.g., dative case (object), genitive case (possessive).

Predicate:
**Linguistics**: the study of the units, nature, structure, and modification of language. Not a language, the difference between the study of a language and linguistics is *abstraction*.

Concepts in linguistics: grammar, word order (SVO, etc.), i.a.

**Case**: indicates the main role of a construct in a sentence, e.g., dative case (object), genitive case (possessive).

**Predicate**: the part of a sentence that expresses what is said of the subject and usually consists of a verb with or without objects.

**Gender**: 
Linguistics: the study of the units, nature, structure, and modification of language. Not a language, the difference between the study of a language and linguistics is abstraction.

Concepts in linguistics: grammar, word order (SVO, etc.), i.a.

Case: indicates the main role of a construct in a sentence, e.g., dative case (object), genitive case (possessive).

Predicate: the part of a sentence that expresses what is said of the subject and usually consists of a verb with or without objects.

Gender: a class of words in a language that may be arbitrary or based on distinguishable characteristics (as shape, social rank, or sex) and determines agreement with and selection of other words.

Agglutination:
Linguistics: the study of the units, nature, structure, and modification of language. Not a language, the difference between the study of a language and linguistics is abstraction.

Concepts in linguistics: grammar, word order (SVO, etc.), i.a.

Case: indicates the main role of a construct in a sentence, e.g., dative case (object), genitive case (possessive).

Predicate: the part of a sentence that expresses what is said of the subject and usually consists of a verb with or without objects.

Gender: a class of words in a language that may be arbitrary or based on distinguishable characteristics (as shape, social rank, or sex) and determines agreement with and selection of other words.

Agglutination: the formation of words by putting together constituents of which each contributes a definite meaning.
Levels of Abstraction

“There is a general distinction concerning thinking: that between categories and individuals, or classes and instances.” page 351.

Thus something as simple as a newspaper might be specified to six levels, as in Douglas R. Hofstadter’s illustration of that ambiguity, with a progression from abstract to concrete in Gödel, Escher, Bach (1979):

1. a publication
2. a newspaper
3. The San Francisco Chronicle
4. the May 18 edition of the Chronicle
5. my copy of the May 18 edition of the Chronicle
6. my copy of the May 18 edition of the Chronicle as it was when I first picked it up (as contrasted with my copy as it was a few days later: in my fireplace, burning)
Abstraction

**Abstraction**: conceptualization without reference to specific instances. The act of determining the fundamental, essential, important, intrinsic properties disassociated from the specific details. A model or representation of something including only the essential properties.

*Software engineering is all about abstraction. Every single concept, construct, and method is entirely abstract. Of course, it doesn’t feel this way to most software engineers. But that’s my point. The main benefit they got from the mathematics they learned in academia was the experience of rigorous reasoning with purely abstract objects and structures.*

Abstraction is the key to computing as it is to mathematics. Both disciplines ask the question: what is the essential essence of stuff?

In computing we are bound by the immutable rules of construction.
Is redundancy essential?
Redundancy in Communication

Natural language is full of redundancy. For example, subject-verb agreement in English and Russian.

- The dog eats. Собака ест.
- The dogs eat. Собаки едят.

But Esperanto and Turkish do not have this redundancy:

- La hundo manĝas. Köpek yer.
- La hundoj manĝas. Köpekler yer.

Redundancy is good for “error correction” in one-time communication, but maybe not for computer programs. It may lead to more possible errors and the need for maintenance problems as programs evolve.

(In Chinese these two statements have a different problem: ambiguity. Gǒ chī. How many dogs eat?)
Abstractions are especially important in Computer Science. Wikipedia even has an article on abstraction just in computer science.

abstraction in cs
As expressed by Benjamin C. Pierce in *Types and Programming Languages* (2002), the abstraction principle reads:

*Each significant piece of functionality in a program should be implemented in just one place in the source code. Where similar functions are carried out by distinct pieces of code, it is generally beneficial to combine them into one by abstracting out the varying parts.*

A key challenge in programming language design is providing mechanisms to abstract out all conceivable commonality.
Software Engineering Dictum

*Don’t Repeat Yourself* (DRY) or *Duplication is Evil* (DIE) is a principle of software development aimed at reducing repetition of information of all kinds. The DRY principle is stated as “Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.” The principle has been formulated by Andy Hunt and Dave Thomas in their book *The Pragmatic Programmer*. They apply it quite broadly to include “database schemas, test plans, the build system, even documentation.”
Practical Tips (Not Our Subject Really)

- Don’t Repeat Yourself (DRY)
- Use Good Names (don’t use “keyboard” for standard input stream)
- Comments Where Needed
- Fail Early
- Avoid Magic Numbers
- One Purpose for Each Variable
- Use white space to Help the Reader
- Don’t Use Global Variables
- Methods Should Return Results, Not Print Them

From MIT Code Review
More programming tips in the *The Pragmatic Programmer* by Hunt and Thomas. A list of the 70 tips at a blog site.
It may be difficult to see the truly great abstractions. But they are fun and great topics to learn about.

- Semi-rings: Dijkstra’s shortest path algorithm or regular expressions
- Monads: the grand unification of imperative and function paradigms (Wadler: “a monad is a monoid in the category of endofunctors, what’s the problem?”)

But abstractions are essential to “ordinary” programming as with parameterized procedures mentioned by Pierce.
Inferring or Learning the Pattern

<table>
<thead>
<tr>
<th>sequence</th>
<th>pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 ...</td>
<td></td>
</tr>
<tr>
<td>2 4 6 8 10 12 ...</td>
<td></td>
</tr>
<tr>
<td>4 9 14 19 24 29 ...</td>
<td></td>
</tr>
<tr>
<td>1 4 9 16 25 36 ...</td>
<td></td>
</tr>
<tr>
<td>1 3 7 15 31 63 ...</td>
<td></td>
</tr>
<tr>
<td>1 2 3 6 11 23 ...</td>
<td></td>
</tr>
</tbody>
</table>

The On-Line Encyclopedia of Integer Sequences

_The Language of Mathematics: Making the Invisible Visible_ and _Mathematics, the Science of Patterns_ by Keith Devlin
### Inferring or Learning the Pattern

<table>
<thead>
<tr>
<th>sequence</th>
<th>pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 ...</td>
<td>( n )</td>
</tr>
<tr>
<td>2 4 6 8 10 12 ...</td>
<td></td>
</tr>
<tr>
<td>4 9 14 19 24 29 ...</td>
<td></td>
</tr>
<tr>
<td>1 4 9 16 25 36 ...</td>
<td></td>
</tr>
<tr>
<td>1 3 7 15 31 63 ...</td>
<td></td>
</tr>
<tr>
<td>1 2 3 6 11 23 ...</td>
<td></td>
</tr>
</tbody>
</table>

The On-Line Encyclopedia of Integer Sequences

_The Language of Mathematics: Making the Invisible Visible_ and _Mathematics, the Science of Patterns_ by Keith Devlin
### Inferring or Learning the Pattern

<table>
<thead>
<tr>
<th>sequence</th>
<th>pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 ...</td>
<td>( n )</td>
</tr>
<tr>
<td>2 4 6 8 10 12 ...</td>
<td>( 2n )</td>
</tr>
<tr>
<td>4 9 14 19 24 29 ...</td>
<td></td>
</tr>
<tr>
<td>1 4 9 16 25 36 ...</td>
<td></td>
</tr>
<tr>
<td>1 3 7 15 31 63 ...</td>
<td></td>
</tr>
<tr>
<td>1 2 3 6 11 23 ...</td>
<td></td>
</tr>
</tbody>
</table>

The On-Line Encyclopedia of Integer Sequences

*The Language of Mathematics: Making the Invisible Visible* and *Mathematics, the Science of Patterns* by Keith Devlin
Inferring or Learning the Pattern

<table>
<thead>
<tr>
<th>sequence</th>
<th>pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 ...</td>
<td>(n)</td>
</tr>
<tr>
<td>2 4 6 8 10 12 ...</td>
<td>(2n)</td>
</tr>
<tr>
<td>4 9 14 19 24 29 ...</td>
<td>(5n - 1)</td>
</tr>
<tr>
<td>1 4 9 16 25 36 ...</td>
<td></td>
</tr>
<tr>
<td>1 3 7 15 31 63 ...</td>
<td></td>
</tr>
<tr>
<td>1 2 3 6 11 23 ...</td>
<td></td>
</tr>
</tbody>
</table>

The On-Line Encyclopedia of Integer Sequences

*The Language of Mathematics: Making the Invisible Visible* and *Mathematics, the Science of Patterns* by Keith Devlin
Inferring or Learning the Pattern

<table>
<thead>
<tr>
<th>sequence</th>
<th>pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 ...</td>
<td>n</td>
</tr>
<tr>
<td>2 4 6 8 10 12 ...</td>
<td>2n</td>
</tr>
<tr>
<td>4 9 14 19 24 29 ...</td>
<td>5n − 1</td>
</tr>
<tr>
<td>1 4 9 16 25 36 ...</td>
<td>n²</td>
</tr>
<tr>
<td>1 3 7 15 31 63 ...</td>
<td></td>
</tr>
<tr>
<td>1 2 3 6 11 23 ...</td>
<td></td>
</tr>
</tbody>
</table>

The On-Line Encyclopedia of Integer Sequences

*The Language of Mathematics: Making the Invisible Visible* and *Mathematics, the Science of Patterns* by Keith Devlin
### Inferring or Learning the Pattern

<table>
<thead>
<tr>
<th>sequence</th>
<th>pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 ...</td>
<td>$n$</td>
</tr>
<tr>
<td>2 4 6 8 10 12 ...</td>
<td>$2n$</td>
</tr>
<tr>
<td>4 9 14 19 24 29 ...</td>
<td>$5n - 1$</td>
</tr>
<tr>
<td>1 4 9 16 25 36 ...</td>
<td>$n^2$</td>
</tr>
<tr>
<td>1 3 7 15 31 63 ...</td>
<td>$2^n - 1$</td>
</tr>
</tbody>
</table>

The On-Line Encyclopedia of Integer Sequences

*The Language of Mathematics: Making the Invisible Visible* and *Mathematics, the Science of Patterns* by Keith Devlin
## Inferring or Learning the Pattern

<table>
<thead>
<tr>
<th>sequence</th>
<th>pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 ...</td>
<td>$n$</td>
</tr>
<tr>
<td>2 4 6 8 10 12 ...</td>
<td>$2n$</td>
</tr>
<tr>
<td>4 9 14 19 24 29 ...</td>
<td>$5n - 1$</td>
</tr>
<tr>
<td>1 4 9 16 25 36 ...</td>
<td>$n^2$</td>
</tr>
<tr>
<td>1 3 7 15 31 63 ...</td>
<td>$2^n - 1$</td>
</tr>
<tr>
<td>1 2 3 6 11 23 ...</td>
<td>trees with $n$ nodes</td>
</tr>
</tbody>
</table>

The On-Line Encyclopedia of Integer Sequences

*The Language of Mathematics: Making the Invisible Visible* and *Mathematics, the Science of Patterns* by Keith Devlin
Making maps is another fascinating example of abstraction in action.

Consider the problem of giving directions from the Rathaus to Karlsplatz in downtown Munich. An aerial photograph may be handy, but ...
In cartography, a *topological map* is a type of diagram that has been simplified so that only vital information remains and unnecessary detail has been removed. These maps lack scale, and distance and direction, but the relationship between points is maintained. A good example is the well-known tube map of the London Underground.
Henry Charles Beck’s 1933 topological pocket map of the London Underground
Popular, innovative, and influential
knowledge is approximately like information
computer science is approximately like information science
The purpose of Unicode is to represent the abstract information content of writing (English, Chinese, etc). Formalizing knowledge is an important step in expressing it in a formal language. In some writing systems this is easier than others. In some cultures writing (calligraphy) is an art form that transcends information.

The dog eats.
Собака ест.

U+0421 CYRILLIC CAPITAL LETTER ES
U+043E CYRILLIC SMALL LETTER O
U+0431 CYRILLIC SMALL LETTER BE
U+043A CYRILLIC SMALL LETTER KA
U+0430 CYRILLIC SMALL LETTER A
U+0020 SPACE
U+0435 CYRILLIC SMALL LETTER IE
U+0441 CYRILLIC SMALL LETTER ES
U+0442 CYRILLIC SMALL LETTER TE
U+002E FULL STOP
In Arabic calligraphy, the Basmala is a prevalent motif. In Unicode, the Basmala is encoded as one symbol at code point:

\[ U+0628 \ ARABIC \ LETTER \ BEH \]
\[ U+0650 \ ARABIC \ KASRA \]
\[ U+0633 \ ARABIC \ LETTER \ SEEN \]
\[ U+0652 \ ARABIC \ SUKUN \]
\[ U+0645 \ ARABIC \ LETTER \ MEEM \]
\[ U+0650 \ ARABIC \ KASRA \]
\[ U+0620 \ SPACE \]
\[ U+0671 \ ARABIC \ LETTER \ ALEF \ WASLA \]
\[ U+0644 \ ARABIC \ LETTER \ LAM \]
\[ U+0644 \ ARABIC \ LETTER \ LAM \]
\[ U+064E \ ARABIC \ FATHA \]
\[ U+0651 \ ARABIC \ SHADDA \]
\[ U+0670 \ ARABIC \ LETTER \ SUPERSCRIPT \ ALEF \]
\[ U+0647 \ ARABIC \ LETTER \ HEH \]
\[ U+0650 \ ARABIC \ KASRA \]
\[ U+0620 \ SPACE \]
\[ U+0671 \ ARABIC \ LETTER \ ALEF \ WASLA \]
\[ U+0644 \ ARABIC \ LETTER \ LAM \]
\[ U+0631 \ ARABIC \ LETTER \ REH \]
\[ U+064E \ ARABIC \ FATHA \]
\[ U+0651 \ ARABIC \ SHADDA \]
\[ U+0660 \ ARABIC \ LETTER \ HAH \]
\[ U+0645 \ ARABIC \ LETTER \ MEEM \]
\[ U+064E \ ARABIC \ FATHA \]
\[ U+0670 \ ARABIC \ LETTER \ SUPERSCRIPT \ ALEF \]
\[ U+0664 \ ARABIC \ LETTER \ NOON \]
\[ U+0671 \ ARABIC \ LETTER \ ALEF \ WASLA \]

The Basmala is the Islamic phrase “In the name of God, the Most Gracious, the Most Merciful.”

\[ bi-smi \ llahi \ r-rahmāni \ r-rahīmi \]

In the name of / Allah / The Beneficent / The Merciful
We now know that electronic technology has no more to contribute to computing than the physical equipment. We now know that a programmable computer is no more and no less than an extremely handy device for realizing any conceivable mechanism without changing a single wire, and that the core challenge for computing science is a conceptual one, viz. what (abstract) mechanisms we can conceive without getting lost in the complexities of our own making.

E. W. Dijkstra, “On a Cultural Gap (Draft),” EWD913, 1985, page 3. [Note Dijsktra started numbering the pages of his EWD’s with 0.]
When abstraction is possible, understanding is possible, when understanding is possible, simplification is possible, when complexity is avoided, high degrees of assuredness is possible.
Ron Ross, the top cybersecurity scientist at the U.S. National Institute of Standards and Technology, on Tuesday told the U.S. Commission on Enhancing National Cybersecurity the coming cybersecurity crisis can only be addressed by building “more trustworthy secure components and systems.” He said it is clear existing security measures are ineffective, given the rising number of successful attacks and breaches despite record cybersecurity investment. “You cannot protect that which you do not understand," Ross said. “Increased complexity translates to increased attack surface."

Example of Complexity: LAMP Stack

Shmuel Csaba Otto Traian [CC BY-SA 3.0], from Wikimedia Commons
Intellectual Complexity

The so-called LAMP stack (Linux OS/Apache internet server/MySQL DBMS/Perl PL) consists of 10 million lines of code, interacting in myriad ways to achieve impressive functionality and performance. This approaches the intellectual complexity of the Saturn V rocket (with three million parts) which took man to the moon. The similarities between these two enormous engineering feats are important: good engineering design practices have been followed; requirements were defined and met; rigorous testing and debugging has taken place. Yet, to date, the LAMP stack is much less well understood than the Saturn V rocket. It is much harder to predict how the LAMP stack will perform under varying conditions and where things might go wrong than it is to consider how the Saturn V may behave in different operating environments.

Morrison and Snodgrass, CACM, 2011
The right abstractions combat complexity, but they take effort.

*It is true that we live in a complex world and strive to solve inherently complex problems, which often do require complex mechanisms. However, this should not diminish our desire for elegant solutions, which convince by their clarity and effectiveness. Simple, elegant solutions are more effective, but they are harder to find than complex ones, and they require more time, which we often believe to be unaffordable.*

Niklaus Wirth, Turing Award lecture, 1984.
Sometimes people make something overly complex on purpose.

There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies; and the other way is to make it so complicated that there are no obvious deficiencies.

Charles Anthony Richard Hoare, Turing Award lecture, 1981
Expression versus Visualization

- Abstractions often lead to a model, e.g., list of longitude, latitude pairs representing the shorelines of the continents.
- Scientific visualization of data (models) is an important branch of computer science, e.g., equal-area and conformal projections are different visualizations of the same data.
- Abstractions, models, visualization are all relative to point of view, i.e., what's important in one context is unimportant to another.
- Visualization is different but related to expression — visualization is “to the mind” and expression “from the mind.” The human mind can perceive graphical information quite readily.
“Artsy” people sometimes prefer visualization; “mathy” people sometimes prefer expression.

Some literature about visualizations:

- A classic: *The Visual Display of Quantitative Information*, by Edward R. Tufte
- *Book of Comparisons*, October, 1980, by The Diagram Group
- *The Functional Art: An introduction to information graphics and visualization* by Alberto Cairo
- *Interactive Data Visualization for the Web* by Scott Murray, March 11, 2013, "O’Reilly Media, Inc." d2js
- *Visualizing Data and The Elements of Graphing Data* by William Cleveland.
Data

Points on a sphere can be represented by the latitude and longitude.

<table>
<thead>
<tr>
<th>Latitude</th>
<th>Longitude</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>38.883</td>
<td>-77.033</td>
<td>Washington, D.C.</td>
</tr>
<tr>
<td>55.750</td>
<td>37.700</td>
<td>Moscow, Russia</td>
</tr>
<tr>
<td>-34.667</td>
<td>-58.500</td>
<td>Buenos Aires, Argentina</td>
</tr>
<tr>
<td>19.400</td>
<td>-99.150</td>
<td>Ciudad de México (Mexico City)</td>
</tr>
<tr>
<td>45.417</td>
<td>-75.717</td>
<td>Ottawa, Canada</td>
</tr>
<tr>
<td>-12.050</td>
<td>-77.050</td>
<td>Lima, Perú</td>
</tr>
<tr>
<td>-33.450</td>
<td>-70.667</td>
<td>Santiago, Chile</td>
</tr>
<tr>
<td>48.867</td>
<td>2.333</td>
<td>Paris, France</td>
</tr>
<tr>
<td>56.883</td>
<td>24.133</td>
<td>Riga, Latvia</td>
</tr>
</tbody>
</table>
Topological View of Earth (rectilinear)

NASA “Blue Marble” image, part of the Visible Earth Catalog
Mercator Projection
Given the longitude east of Greenwich $\lambda$ and the latitude $\phi$ north of the equator, the underlying model of the Mercator projection is given by:

\[
x = R\lambda \\
y = R \ln \tan\left(\frac{\pi}{4} + \frac{\phi}{2}\right)
\]
Behrmann Cylindrical Equal-Area
Cylindrical Equal-Area

The underlying models of cylindrical equal-area projections:

\[ x = R(\lambda - \lambda_0) \cos \phi_s \]
\[ y = R \sin \phi \sec \phi_s \]

<table>
<thead>
<tr>
<th>( \phi_s )</th>
<th>map projection</th>
</tr>
</thead>
<tbody>
<tr>
<td>0°</td>
<td>Lambert</td>
</tr>
<tr>
<td>30°</td>
<td>Behrman</td>
</tr>
<tr>
<td>37.400°</td>
<td>Trystan Edwards (or 37.383)</td>
</tr>
<tr>
<td>44.138°</td>
<td>Peters (or 45)</td>
</tr>
<tr>
<td>45°</td>
<td>Gall Orthographic</td>
</tr>
<tr>
<td>50°</td>
<td>Balthasart</td>
</tr>
</tbody>
</table>
Mollweide
The underlying model of the Mollweide projection:

\[ x = 2\sqrt{2}R(\lambda - \lambda_0)(\cos \theta)/\pi \]

\[ y = \sqrt{2}R \sin \theta \]

where \( \theta \) is defined for a given \( \phi \) by:

\[ 2\theta + \sin 2\theta = \pi \sin \phi \]
### Definition (Abstraction)

Conceptualization without reference to specific instances. The act of determining the fundamental, essential, important, intrinsic properties disassociated from the specific details.

### Definition (Model)

A formal or schematic description of a system, theory, phenomenon.

### Definition (Visualization)

The act of making a visible presentation of data, particularly a graphical presentation. For example, a graph is a visualization of a function. Virtual reality is another form of visualization.
<table>
<thead>
<tr>
<th>Definition (Expression)</th>
<th>representing in a medium (as words or programs) or representing in some medium</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition (To represent)</td>
<td>to portray, exhibit, form an image, to describe as having a specified character or quality, to apprehend by means of an idea</td>
</tr>
<tr>
<td>Definition (To denote something)</td>
<td>to find expression in a language of some notion</td>
</tr>
<tr>
<td>Definition (Medium)</td>
<td>the means of conveying or transmitting something;</td>
</tr>
<tr>
<td>Definition (Language)</td>
<td>System of symbols, signs, sounds, or gestures used for communication</td>
</tr>
<tr>
<td>Definition (Notation)</td>
<td>Marks, signs, figures, or characters representing something.</td>
</tr>
<tr>
<td>Definition (Complex)</td>
<td>hard to separate, analyze, or solve</td>
</tr>
<tr>
<td>Definition (Simple)</td>
<td>readily understood or apprehended (by a person)</td>
</tr>
<tr>
<td>Definition (Natural)</td>
<td>obvious (to a person), expected and unaffected</td>
</tr>
</tbody>
</table>
The goal of the field of programming languages is to find natural (to people) expression of computation which is efficiently translatable to computers.

What is natural is shaped by psychology, experience, and education.

Progress: better notation enables better abstraction, and *vice versa*.
Examples of the power of a well-contrived notation to condense into small space a meaning which would—in ordinary language—require several lines, or even pages, can hardly have escaped the notice of most of my readers: . . . instead of creating any obscurity, the expressions are far more readily understood than if they were written at length.

Charles Babbage, 1827.

- Multiplication in Roman numerals
- FORTRAN, rocket launch
- C break statement and the AT&T telephone outage
Charles Babbage (1791–1871)

Babbage was one of the first to have a modern vision of a calculating machine:

I was sitting in the rooms of the Analytical Society, at Cambridge, my head leaning forward on the table in a kind of dreamy mood, with a table of logarithms lying open before me. Another member, coming into the room, and seeing me half asleep, called out, “Well, Babbage, what are you dreaming about?” to which I replied “I am thinking that all these tables” (pointing to the logarithms) “might be calculated by machinery.”
Value of Notation

By relieving the brain of all unnecessary work, a good notation sets it free to concentrate on more advanced problems, and in effect increases the mental power of the race. Before the introduction of the Arabic notation, multiplication was difficult, and the division even of integers called into play the highest mathematical faculties. Probably nothing in the modern world would have more astonished a Greek mathematician than to learn that... a large proportion of the population of Western Europe could perform the operation of division for the largest numbers. This fact would have seemed to him a sheer impossibility. ... Our modern power of easy reckoning with decimal fractions is the almost miraculous result of the gradual discovery of a perfect notation.

purposes, however, the history of the notation is a detail. The interesting point to notice is the admirable illustration which this numeral system affords of the enormous importance of a good notation. By relieving the brain of all unnecessary work, a good notation sets it free to concentrate on more advanced problems, and in effect increases the mental power of the race. Before the introduction of the Arabic notation, multiplication was difficult, and the division even of integers called into play the highest mathematical faculties. Probably nothing in the modern world would have more astonished a Greek mathematician than to learn that, under the influence of compulsory education, the whole population of Western Europe, from the highest to the lowest, could perform the operation of division for the largest numbers. This fact would have seemed to him a sheer impossibility. The consequential extension of the notation to decimal fractions was not accomplished till the seventeenth century. Our modern power of easy reckoning with decimal fractions is the almost miraculous result of the gradual discovery of a perfect notation.

Mathematics is often considered a difficult and mysterious science, because of the numerous symbols which it employs. Of course, nothing is more incomprehensible than
Sapir-Whorf Hypothesis

The medium constrains expressiveness.

This is known as the Sapir-Whorf hypothesis named after Edward Sapir (1884-1939), the American (Pomeranian-born) anthropologist and linguist; a founder of ethnolinguistics and principal developer of American school of structural linguistics.
Sapir-Whorf Hypothesis

We cut nature up, organize it into concepts, and ascribe significances as we do, largely because we are parties to an agreement to organize it in this way—an agreement that holds throughout our speech community and is codified in patterns of our languages. The agreement is, of course, an implicit and unstated one, but its terms are absolutely obligatory; we cannot talk at all except by subscribing to the organization and classification of data which the agreement decrees.

Figure 10. Languages classify items of experience differently. The class corresponding to one word and one thought in language A may be regarded by language B as two or more classes corresponding to two or more words and thoughts.
Die Grenze meiner Sprache bedeuten die Grenzen meiner Welt – Tractus Logico-Philosophicus 5.6
Die Sprache bestimmt das Denken.
He who controls vocabulary controls thought.

Ludwig Wittgenstein (1889–1951)

The end of Tractus:
Wovon man nicht sprechen kann, darüber muss man schweigen
We do not learn language by reading a dictionary, and we do not think or speak in terms of dictionary definitions. Meaning is always more fluid. Nevertheless, we are hemmed in, even trapped, by common usage. Senses we wish to evade entrap us. The greatest escape route is not only humor, but poetry, or art in general. Art does not, of course, liberate us completely from meaning, but it gives a certain measure of freedom, provides elbow room.

Charles Rosen, the pianist who died in 2012, in an essay entitled “Freedom and Art.”
Names of Colors

Can you perceive a color if you do not have a name for it?

- Himba tribe can’t see blue

Lest one feels superior, we do not perceive shades of green that they can.
A researcher named Jules Davidoff traveled to Namibia to investigate this, where he conducted an experiment with the Himba tribe, which speaks a language that has no word for blue or distinction between blue and green. When shown a circle with 11 green squares and one blue, they could not pick out which one was different from the others — or those who could see a difference took much longer and made more mistakes than would make sense to us, who can clearly spot the blue square.

But the Himba have more words for types of green than we do in English. When looking at a circle of green squares with only one slightly different shade, they could immediately spot the different one.

Kevin Loria, “No one could see the color blue until modern times”, Business Insider 2/27/2015

See Radiolab 21 May 2012
Can you find the blue square?
Which green one is different?
The middle one in the row is supposed to be different (but electronic devices do not communicate and display colors accurately).
Whether or not the Shapir-Worf hypothesis holds in human communication (probably not), in programming, the lack of experience in the diversity of expression in an impediment.

It is easy to understand why this happens.
In making our adjustments to our particular programming language, we can easily become attached to it simply because we now have too much invested in it. Most people would prefer almost any amount of pain to giving up the familiarity of some constant companion for an unknown quantity. We see this effect when we try to teach a programmer his second language. Teaching the first is no great problem, for he has no investment in any other. By the time he has learned two or more, he is aware that more things exist in this world than he has dreamed of. But letting go of the first is, to him, just a promise of pain with no promise of compensating pleasure.

The Psychology of Computer Programming by Gerald M. Weinberg
Enough analogy with linguistics and geography, what are the important abstractions in computation? (The really big ones.)

Why do we care? The abstractions in computing that give rise to useful mediums of expression are clearly useful for designing programming languages. Sometimes (most of the time) we see these abstractions clearly only in retrospect. One way of organizing these abstractions gives rise to three kinds of programming languages. Often we call these programming language paradigms.
Optional Digression: What is meant by the word *paradigm*? It is an important and complex word which people use differently.
Thomas Samuel Kuhn (1922–1996)

He received a Ph.D. in physics from Harvard University in 1949. In 1964, he was named M. Taylor Pyne Professor of Philosophy and History of Science at Princeton University. In 1983 he was named Laurence S. Rockefeller Professor of Philosophy at MIT. Kuhn’s most renown work is *The Structure of Scientific Revolutions*, which he wrote while a graduate student in theoretical physics at Harvard.
Thomas Samuel Kuhn (1922–1996)

Kuhn was responsible for popularizing the term paradigm, which he described as essentially a collection of beliefs shared by scientists, a set of agreements about how problems are to be understood. According to Kuhn, paradigms are essential to scientific inquiry, for “no natural history can be interpreted in the absence of at least some implicit body of intertwined theoretical and methodological belief that permits selection, evaluation, and criticism.” Indeed, a paradigm guides the research efforts of scientific communities, and it is this criterion that most clearly identifies a field as a science. A fundamental theme of Kuhn’s argument is that the typical developmental pattern of a mature science is the successive transition from one paradigm to another through a process of revolution. When a paradigm shift takes place, “a scientist’s world is qualitatively transformed [and] quantitatively enriched by fundamental novelties of either fact or theory.”

From the *New York Times* Obituary.
The word “paradigm” comes from the Greek word \( \pi\alpha\rho\alpha\delta\epsilon\gamma\mu\alpha \) (“\textit{paradeigma}”) meaning example, instance, model, or pattern.

The common English definitions for the word “paradigm” are:

1. An example or pattern; especially: an outstandingly clear or typical example or archetype.
2. \textit{Grammar}: a set of all inflected forms based on a single stem.
3. \textit{Philosophical or theoretical framework in which laws and generalizations are formulated}.
4. A collection of beliefs or agreements shared by scientists.
Ada and C

procedure GCD(A:in out Natural;B:Natural) is
begin
    while A /= B loop
        if A>B then
            A := A - B;
        else
            B := B - A;
        end if;
    end loop;
end GCD;

unsigned gcd (unsigned a, unsigned b) {
    while (a != b) {
        if (a>b) a -= b;
        else b -= a;
    }
    return a;
}
let rec gcd a b =
    if a=b then a
    else if a>b then gcd b (a-b)
    else gcd a (b-a)

gcd(A,B,C) :- A=B, G=A.
gcd(A,B,C) :- A>B, C is A-B, gcd(C,B,G).
gcd(A,B,C) :- B>A, C is B-A, gcd(C,A,G).
Computational Paradigms

- Imperative: :=, goto, if
- Functional: beta reduction

\[(\lambda x.b)a \rightarrow b[x := a]\]

- Logic: resolution principle

\[
\frac{Q_1, \ldots, Q_n \leftarrow P_1, P_2, \ldots, P_k \quad R_1, R_2, \ldots, R_m \leftarrow S_1 \ldots, S_l}{(Q_1, \ldots, Q_n, R_2, \ldots, R_m)\sigma \leftarrow (P_2, \ldots, P_k, S_1, \ldots, S_n)\sigma}
\]

Other: SETL (vaguely ZF set theory), APL (combinators), Thue (string rewriting), O'Donnell’s equational logic as a programming language
Conventional programming languages (such as FORTRAN, COBOL, ALGOL 60, ALGOL 68, Pascal, C, Clu, Modula, and Ada) make essential use of assignment statements as the basic construct, around which are build the control abstractions of sequencing, branching and looping, etc. Such languages are called imperative or procedural because of the way programs are based on the idea of instructions to be carried out like a recipe. These instructions incrementally transform a store (made up of cells) by updating the contexts of the cells to achieve some overall effect.

Reade, 1989, page 2
Computational Paradigms: Imperative

A programming language that is characterized by these three properties—the sequential execution of the instructions, the use of variables representing memory locations, and the use of assignment to change the values of variables—is called an imperative language, because its primary feature is a sequence of statements that represent commands, or imperatives.

Louden and Lambert, 3rd, 2012, page 15
Imperative languages are characterized as having an implicit state that is modified (i.e., side effected) by constructs (i.e., commands) in the source language. As a result, such languages generally have a notion of sequencing (of the commands) to permit precise and deterministic control over the state. Most, including the most popular, languages in existence today are imperative.

ACM Computing Surveys, Vol. 21, No. 3, September 1989

To this must be mentioned Python, Java, C++. This despite “lambdas” in these languages. (Mostly Wrong History: “Lambdas are relegated to relative obscurity until Java makes them popular by not having them.”)

These languages support the imperative paradigm as well: LISP (SETQ), Scheme (set!), SETL (:=), ML (:=), APL (<-), and F# (<-),
Computation is the most general purpose a programming language has, but other features are important, too.

- Declarative programming – what versus how
- Object-oriented programming – organization of data
- Structured programming – style, discipline
- Distributed computing – external interaction
- Parallel programming – implementation
- Database programming – persistence of data
And there are still other ways to organize programming languages.

- Scripting programming languages
- Dataflow (e.g., Lucid)
- Constraint Programming
- Visual
- Pedagogic
- Esoteric programming languages ("esolang"), eg., INTERCAL, Whitespace, Thue
Scripting programming languages

Scott: “most actual *uses* of computers require the coordination of multiple programs.” Cf. Scott, 2nd, Chapter 13: Scripting Languages.

ALSU: “interpreted languages with high-level operators designed for ‘gluing together’ computations.”

Ousterhout: “Scripting languages are designed for ‘gluing’ applications; they use typeless approaches to achieve a higher level of programming and more rapid application development.”

Language is not about features (vocabulary), is it about gluing operations?
Compare with Sebesta Section 1.5 Language Categories.
Compare with Sebesta Section 1.5 Language Categories.

11th, page 20. 1.5 Language Categories:
“four bins: imperative, functional, logic, and object oriented.”