

“Computational Thinking will be a fundamental skill used by everyone in the world in the middle of the 21st century”

J.M. Wing, "Computational Thinking," Communications of the ACM Viewpoint, Mar 2006, pp. 33-35.

“Computational Thinking will be a fundamental skill used by everyone in the world in the middle of the 21st century”

J.M. Wing, "Computational Thinking," Communications of the ACM Viewpoint, Mar 2006, pp. 33-35.

- In many areas, it is already a reality: engineers, biologists, chemists, physicist. . . , anthropologist, psychologists, economist, criminologist

“Computational Thinking will be a fundamental skill used by everyone in the world in the middle of the 21st century”

J.M. Wing, "Computational Thinking," Communications of the ACM Viewpoint, Mar 2006, pp. 33-35.

- In many areas, it is already a reality: engineers, biologists, chemists, physicist. . . , anthropologist, psychologists, economist, criminologist
- Google “Computational X” – replace X with your favorite discipline

“Computational Thinking will be a fundamental skill used by everyone in the world in the middle of the 21st century”

J.M. Wing, "Computational Thinking," Communications of the ACM Viewpoint, Mar 2006, pp. 33-35.

- In many areas, it is already a reality: engineers, biologists, chemists, physicist. . . , anthropologist, psychologists, economist, criminologist
- Google “Computational X” – replace X with your favorite discipline
- In every day life, it will become a reality.

What is computational thinking?

What is computational thinking?

Computational thinking is the key

- for solving problems

What is computational thinking?

Computational thinking is the key

- for solving problems
- for designing systems

What is computational thinking?

Computational thinking is the key

- for solving problems
- for designing systems
- for achieving what one human alone cannot do

What is computational thinking?

Computational thinking is the key

- for solving problems
- for designing systems
- for achieving what one human alone cannot do
- for understanding the power and limits of human intelligence and capabilities of machines

What is computational thinking?

Computational thinking is the key

- for solving problems
- for designing systems
- for achieving what one human alone cannot do
- for understanding the power and limits of human intelligence and capabilities of machines

What it is and what it is not

- Conceptual – not just about writing programs
- A way humans think – not computers
- Fundamental – not rote, mechanical skill

What is computational thinking?

Computational thinking is the key

- for solving problems
- for designing systems
- for achieving what one human alone cannot do
- for understanding the power and limits of human intelligence and capabilities of machines

What it is and what it is not

- Conceptual – not just about writing programs
- A way humans think – not computers
- Fundamental – not rote, mechanical skill

Computational thinking is drawing fundamentally on concepts from computer science!

The two A's of computational thinking

- Abstraction
 - Helps solve problems
 - Multiple layers of abstraction
 - Define relationship between abstractions
 - Transfer and solve multiple problems
- Automation
 - Mechanizing the abstraction layers and their relationships
 - Goal is to execute, simulate, observe behavior of abstract problem descriptions

The two A's of computational thinking

- Abstraction
 - Helps solve problems
 - Multiple layers of abstraction
 - Define relationship between abstractions
 - Transfer and solve multiple problems
- Automation
 - Mechanizing the abstraction layers and their relationships
 - Goal is to execute, simulate, observe behavior of abstract problem descriptions

Computer Science is a science of abstraction – creating the right model for a problem and devising the appropriate mechanizable techniques to solve it.

- A. Aho and J. Ullman

The two A's of computational thinking

- Abstraction
 - Helps solve problems
 - Multiple layers of abstraction
 - Define relationship between abstractions
 - Transfer and solve multiple problems
- Automation
 - Mechanizing the abstraction layers and their relationships
 - Goal is to execute, simulate, observe behavior of abstract problem descriptions

Computer Science is a science of abstraction – creating the right model for a problem and devising the appropriate mechanizable techniques to solve it.

- A. Aho and J. Ullman

Example, please !

Search me in the phone book

Search me in the phone book

Lessons learned

- We need a way to describe what we did
- We can apply our solution to other search problems
- There are different solutions; which is the best one?

Representations of computational thinking

Purpose :

- Document solutions (to be able to revisit later)
- Communicate solutions (to your friend, co-worker, etc.)
- Compare solutions (Which one is better? Why? Equally powerful?)
- Implement solutions

Representations of computational thinking

Purpose :

- Document solutions (to be able to revisit later)
- Communicate solutions (to your friend, co-worker, etc.)
- Compare solutions (Which one is better? Why? Equally powerful?)
- Implement solutions

Representations of computational thinking

Purpose :

- Document solutions (to be able to revisit later)
- Communicate solutions (to your friend, co-worker, etc.)
- Compare solutions (Which one is better? Why? Equally powerful?)
- Implement solutions

This talk:

- From flow charts to abstract machines
- Thinking about the limits and power of computation
- Excursions: computational thinking in ancient and modern times
- (Programming) Languages for computations
-

“We can look at the current field of problem solving by computer as a series of ideas about how to represent a problem. If a problem can be cast into one of these representations in a natural way, then it is possible to manipulate it and stand some chance of solving it”

From flow charts to abstract machines

Flow charts : where they come from – what they are today

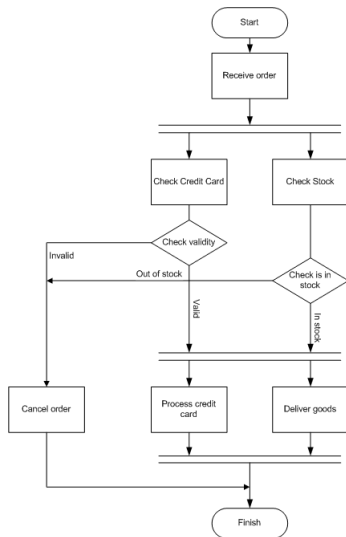
Online order system

When did flow charts originate?

- 1921: F. Gilberth introduces it to the American Society of Mechanical Engineers (ASME)
- 1930s: Industrial engineering curricula
- 1940s: Reaches industry
- 1950s: Model computer programs

What is their use today?

- Unified Modelling Language (UML)
- Used pervasively in software engineering.

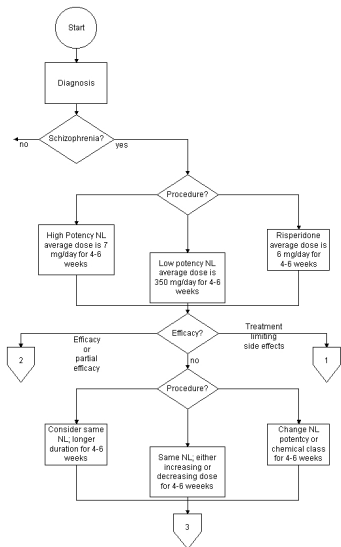


Flow charts in other sciences

Used commonly in other science

- Medicine
- Chemistry
- Biology
- Economics
- ...

Diagnosis for Schizophrenia



Flow charts can be complex!

Advantages

- Organize the thoughts
- High-level view; abstract over details;

Flow charts can be complex!

Advantages

- Organize the thoughts
- High-level view; abstract over details;

Disadvantages

- Interactions can be complex!

Flow charts can be complex!

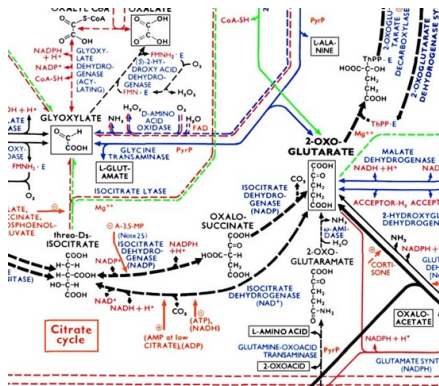
Advantages

- Organize the thoughts
- High-level view; abstract over details;

Disadvantages

- Interactions can be complex!
- No rigorous analysis possible
What is the shortest path? What components are connected?
- Hard to communicate the meaning?
What is the meaning of each line?
What is the meaning of the colors?

System biology



Flow charts can be complex!

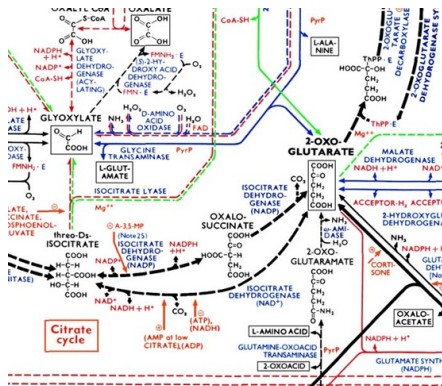
Advantages

- Organize the thoughts
- High-level view; abstract over details;

Disadvantages

- Interactions can be complex!
- No rigorous analysis possible
What is the shortest path? What components are connected?
- Hard to communicate the meaning?
What is the meaning of each line?
What is the meaning of the colors?

System biology



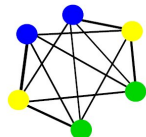
"The limits of my language mean the limits of my world."

Ludwig Wittgenstein

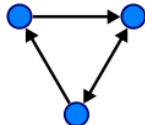
Graphs

- Abstract representation of a set of objects where some pairs of the objects are connected by links.

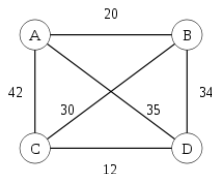
Undirected graph



Directed graph



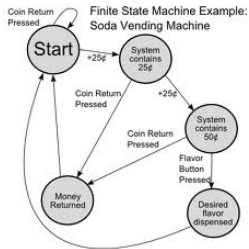
Weighted graph



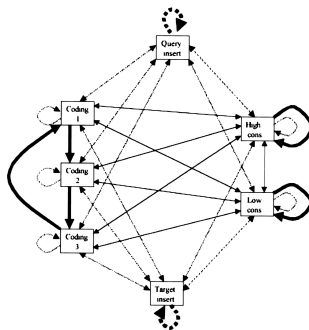
- Some questions about graphs:
 - What is the shortest path?
 - Can we partition graphs in strongly connected components?
 - Optimal route through a graph connecting two nodes?
 - Can we color the nodes such that directly connected nodes will have different colors?

Finite state machine

- Similar to flow charts, but more precise, mathematical model
- Composed of a finite number of states, transitions between those states, and actions
- Applications: hardware design, protocol design, biological and neurological systems, linguistics



Finite state machine depiction of seven state aligner.



Kent W J, Zahler A M Genome Res. 2000;10:1115-1125

Abstract machines and computability

- **Abstract machines** are a theoretical model of a computer and allow us to describe our problem computationally

- Finite State Machines / Automata

- Turing Machines

- Post Machines

- Register machines

- Markov algorithms

- ...

} (theoretical) languages
for computations!

⇒ Amazing fact: These are all formally equivalent!

⇒ Whether you have a Mac or a PC they can compute the same things (theoretically) and are subject to the same limitations.

Church-Turing thesis:

Abstract machines capture the informal notion of computability.

What language and representation we choose depends on what questions we try to answer!

What language and representation we choose depends on what questions we try to answer!

Thinking about the limits and power of computation

- What does it mean to compute?
- What are the limitations of computations?
- Can we solve any problem?
- Can we solve any problem **efficiently**, i.e. in practice?

What does it mean to compute?

What does it mean to compute?

- Machine-like descriptions : Abstract machines
 - Various formal models of computability:
 - μ -recursive functions (Gödel 1930s)
 - The λ -calculus (Church 1936): e. g., $(\lambda x. 2x)$
 - Logic: Modus ponens with axioms about arithmetic (1930s)
- ⇒ Even more amazing fact: These are all formally equivalent!

Limitations of computing

- Does there exist a yes-or-no answer for every problem?
 - ⇒ Turing (1936) Halting problem
 - ⇒ Church (1936) Decision problem
- Are problems which cannot be computed?

Limitations of computing

- Does there exist a yes-or-no answer for every problem?
 - ⇒ Turing (1936) Halting problem
 - ⇒ Church (1936) Decision problem
- Are problems which cannot be computed? Yes!

Limitations of computing

- Does there exist a yes-or-no answer for every problem?
 - ⇒ Turing (1936) Halting problem
 - ⇒ Church (1936) Decision problem
- Are problems which cannot be computed? Yes!
- These are formal *limitations* of *all* models of computability (even your PC or MacBook!)

Limitations of computing

- Does there exist a yes-or-no answer for every problem?
 - ⇒ Turing (1936) Halting problem
 - ⇒ Church (1936) Decision problem
- Are problems which cannot be computed? Yes!
- These are formal *limitations* of *all* models of computability (even your PC or MacBook!)
- Suppose that solutions to a problem can be verified quickly. Then, can the solutions themselves also be computed quickly?

Limitations of computing

- Does there exist a yes-or-no answer for every problem?
 - ⇒ Turing (1936) Halting problem
 - ⇒ Church (1936) Decision problem
- Are problems which cannot be computed? Yes!
- These are formal *limitations* of *all* models of computability (even your PC or MacBook!)
- Suppose that solutions to a problem can be verified quickly. Then, can the solutions themselves also be computed quickly?
 - Stipulated by S. Cook in 1971 – as of today: unsolved
 - 1 Million Dollar prize for a solution offered by the Clay Institute of Mathematics!

\$1 M Question: $P = NP$?

Example NP-Problem:

Suppose that you are organizing housing accommodations for a group of 400 students. Space is limited and only 100 students will receive places in the dormitory. To complicate matters, the Dean has provided you with a list of pairs of incompatible students, and requested that no pair from this list appear in your final choice.

- Easy to check if solution is satisfactory
- Generating a solution is hard – it is completely impractical!

\$1 M Question: $P = NP$?

Example NP-Problem:

Suppose that you are organizing housing accommodations for a group of 400 students. Space is limited and only 100 students will receive places in the dormitory. To complicate matters, the Dean has provided you with a list of pairs of incompatible students, and requested that no pair from this list appear in your final choice.

- Easy to check if solution is satisfactory
- Generating a solution is hard – it is completely impractical!
N = combinations of choosing 100 students from 400 applicants ;
N > number of atoms in the universe

\$1 M Question: $P = NP$?

Example NP-Problem:

Suppose that you are organizing housing accommodations for a group of 400 students. Space is limited and only 100 students will receive places in the dormitory. To complicate matters, the Dean has provided you with a list of pairs of incompatible students, and requested that no pair from this list appear in your final choice.

- Easy to check if solution is satisfactory
- Generating a solution is hard – it is completely impractical!
N = combinations of choosing 100 students from 400 applicants ;
N > number of atoms in the universe
- The problem cannot be solved by brute force

\$1 M Question: $P = NP$?

Example NP-Problem:

Suppose that you are organizing housing accommodations for a group of 400 students. Space is limited and only 100 students will receive places in the dormitory. To complicate matters, the Dean has provided you with a list of pairs of incompatible students, and requested that no pair from this list appear in your final choice.

- Easy to check if solution is satisfactory
- Generating a solution is hard – it is completely impractical!
N = combinations of choosing 100 students from 400 applicants ;
N > number of atoms in the universe
- The problem cannot be solved by brute force – but can we compute a solution quickly by some procedure?

Consequences of $P = NP$

Negative consequences:

- Cryptography: We rely on certain problems being difficult; A constructive, efficient solution could break many existing cryptosystems such as Public-key cryptography which is used in transactions with banks, online shopping sites, etc.

Positive consequences (enormous!):

- Rendering tractable many currently mathematically intractable problems.
- Some NP problems: Travelling salesman problem, logistics, protein structure prediction,

“...it would transform mathematics by allowing a computer to find a formal proof of any theorem which has a proof of a reasonable length, since formal proofs can easily be recognized in polynomial time. Example problems may well include all of the CMI prize problems.”

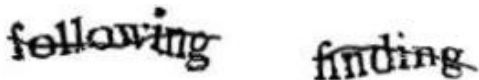
Stephen Cook

Beyond the computer as a machine

What is computable when we consider the computer as the combination of human and machine?

Captcha : Completely Automated Public Turing test to tell Computers and Humans Apart

- Challenge-response test used in computing to ensure that the response is not generated by a computer.



following finding

- Computer asks the user to complete a simple test which the computer is able to generate and grade.
- Because other computers are unable to solve the CAPTCHA, any user entering a correct solution is presumed to be human.
- **Reverse Turing test**: Turing test is a test of a machine's ability to demonstrate intelligence.

Excursion: Computational thinking in ancient and modern times

A look in the past

Long before there were computers, people were interested in describing computations and thinking computationally.

- Diophantus of Alexandria (ca. 210–285)
 - *Arithmetica*, collection of problems and arithmetical solutions
- Muhammad ibn Musa al-Khwarizmi (ca. 780–850)
 - Persian mathematician, astronomer and geographer
 - *Algebra*, systematic solution of linear and quadratic equations (→ *algorithm*)
- François Viète (1540–1603)
 - popularized use of variables for unknowns in algebraic equations
- René Descartes (1596–1650)
 - Introduced ‘cartesian’ coordinates to translate problems in geometry to algebra (‘analytic geometry’)
 - Very powerful method: Can apply all the machinery of algebra to solve geometric problems!

The language of logic

- Gottfried Wilhelm Leibniz (1646–1716) : Philosopher, mathematician (inventor of calculus), built calculating machine, binary arithmetic
- Leibniz's dream:
Compile an *encyclopedia* of all human knowledge.
- a) Determine the key notions and select symbols for them: *characteristica universalis*.
- b) Formulate *rules of deduction* for manipulating these symbols: *calculus ratiocinator*.

'The only way to rectify our reasonings is to make them as tangible as those of the Mathematicians, so that we can find our error at a glance, and when there are disputes among persons, we can simply say: Let us calculate [calculemus], without further ado, to see who is right.' (The Art of Discovery 1685, W 51)

Propositional and predicate logic

- George Boole (1815–1864) : **Propositional logic**
Logical reasoning \leftrightarrow algebraic reasoning! Formulate the laws of reasoning about classes (concepts) in analogy to the rules of algebra
- Gottlob Frege (1848–1925) : **Predicate logic**
 - Boole's logic is not expressive enough.
 - We need to express (quantifiers):
'For all x , ...' and *'There exists an x , such that ...'*
- Notation that allows to make proofs gap-free: mechanical, without recourse to intuition
 - E. g., *'All cats are furry.'*
'Everything that is a cat is also furry.'
'For all x , if x is a cat then x is furry.'
 - Modern notation: $\forall x. (Cat(x) \rightarrow Furry(x)).$

Logic - the calculus of computer science

- Unifying foundational framework
- Powerful tool for modeling and reasoning about aspects of computation i.e. correctness
- Translate state transition systems into logic!
- Applications in computer science
 - Reasoning and verifying computer systems
 - Query language for database systems
 - Describing complexity of a system
 - Programs as proofs; Propositions as types

"I expect that digital computing machines will eventually stimulate a considerable interest in symbolic logic ... The language in which one communicates with these machines .. forms a sort of symbolic logic."

A. Turing

Programming languages for computations

Programming language paradigms

- Turing machines → **imperative programming**
 - Series of instructions and commands; loops
 - Maintain state implicitly or explicitly
 - Examples: Assembler, C, Basic, C++, Java
- λ -calculus → **functional programming**
 - Avoid state and persistent data
 - Driving force: Recursion, functions, data-types
 - Computation = application of functions to arguments
 - Examples: Lisp, ML, Haskell, F#
- Logic → **logic programming**
 - No state
 - Driving force: Recursion, Subset of first-order logic
 - Computation = proof search
 - Examples: Prolog, Datalog, logical frameworks

“A language that doesn't affect the way you think about programming, is not worth knowing.”

- Alan Perlis

There are many programming languages

A+ A++ A# .NET A# (Axiom) A-0 ABAP ABC ABC ALGOL ABLE ABSET ABSYS ACC Accent ActionScript Ace DASL ACT-III Ada ALGOL APL Ajax AppleScript AMOS Arc AutoHotkey Autolt AWK AMZI Adobe Flex ASP.NET B B BACI Baja BASIC bc bcompile BCPL BeanShell BETA Bigwig Big Snake Bistro BLISS Blitz Basic blitz max blitz plus blitz 3d= Block And List Manipulation (BALM) Blue - Rejected prototype for Ada Blue Boo Bourne shell Bourne-Again shell Boxx BPEL Brainfuck BUGSYS BuildProfessional BYOND C C C- C-script C++ # C shell (csh) Caché ObjectScript Caml Cat Cayenne C-BOT Cecil Cesil CFML Cg Ch interpreter Chapel CHAIN Charity Chef Chey CHILL CHIP-8 chomski Chrome Chuck Cilk CICS CL Clarion Clean Clipper CLIST Clojure CLU CMS-2 COBOL o CobolScript Cobra CODE ColdFusion COMAL Common Intermediate Language (CIL) Common Lisp Component Pascal COMMIT Concept Concurrent Clean Constraint Handling Rules Converge CORAL66 Corn CorVision COWSEL CPL CSP CSS Csound Cue Curl Curry Cyclone D D Dao DASL DASL DarkBASIC DarkBASIC Professional Dataflex Datalog dBASE dc Deesel Delphi Dialect DinkC DCL Dialog Manager DIBOL DL/I Dream Maker Dylan Dynace E E Ease EASY Easy PL/I EASYTRIEVE PLUS eC ECMAScript eDeveloper Edinburgh IMP Einstein Eiffel Elan elastiC Elf Emacs Lisp EGL Programming Language (EGL) Epigram Erlang Escapade Esterel Euclid Euphoria Euler EXEC EXEC2 F F F# Factor Falcon Fan Felix Ferite F# Fjölfnir FL FLOW-MATIC FOCAL FOCUS FOIL FORMAC Formula language Forth Fortran Fortress FoxPro FP Frag Script Franz Lisp Frink Frontier F-Script Fuxi Programming Language G GAAMIL GM GAP Gambas Game Maker Language G-code General Algebraic Modeling System Generic Java Gibiane G (LabVIEW) Gödel Godiva GOTRAN GOTO++ GPSS GraphTalk GRASS Green Green Groovy H HAL/S HAScript Haskell HaXe High Level Assembly Hop HTML HyperTalk I IBM Basic assembly language IBM RPG ICI Icon IDL IKE IMP Inform Information Processing Language (IPL) Informix-4GL Io IPTSCRAE Interactive System Productivity Facility Internet Relay Chat (IRC) J J J# J++ JADE JAG Jal Janus JASS Java JavaScript Jim++ JCL Join Java JOSS Joule JOVIAL Joy JScript JSP Java EE Java ME Joomla! K K KEE Kiev Korn Shell KIF Kite Kogut KRC KRL KRYPTON L L LabVIEW Ladder Lagoon LANSa Lasso Lava Leda Lead Leadwerks Script Legoscript Lexico Liberty BASIC Limbo Limnor LINC Lingo LISA Lisaac Lisp Logo LOLCODE LPC LSL LSE Lua Lucid Lush Lustre LYaPAS M M4 MacRuby MAD Magik Magma MapBasic Maple MAPPER MARK-IV Mary MASM Microsoft Assembly x86 Mathematica MATLAB Maxima MaxScript internal language 3D Studio Max Maya (MEL) Mercury Mesa Michigan Algorithm Decoder Microcode MicroScript MillScript MIMIC Mindscript Miranda MIVA Script ML Moby Model 204 User Language Modula Modula-2 Modula-3 Mondrian Mortran Moto MOUSE MSIL MSL MUMPS N Napier88 Natural NEAT Nemerle NESL Net.Data NewLISP NewtonScript NGL Nial Lego Mindstorms NXT (NXT-G) Nice Nickle Noscica NQC NXC Nu O o:XML Oberon Object Lisp ObjectLOGO Object Pascal Objective-C Objective-J Objective Caml Obliq Ocaml occam occam-π Octave OmniMark Opal Open programming language OPL5 ORCA/Modula-2 Organiser Programming Language (OPL) Oxygene Oz P PARI/GP Parser Pascal Pawn PBASIC PCASTL PEARL Perl PHP Prologram Pico Piet Pike PIKT PILOT Pizma PL/0 PL/8 PL/B PL/C PL/I PL/M PL/P PL/SQL Plankalkül PLEXIL Pliant PPL POP-11 Poplog PostScript Processing Prograph Progress 4GL Prolog Promela Protheus PRO-IV ProvideX Python Q Q Qi QtScript QuakeC QPL R R R++ Ratfiv Ratfor RBScript rc REBOL Redcode REFAL Reia Reilly Revolution REXX Rlab Robot Scripting Language (RSL) RPG RPL Ruby REALbasic S S S2 S-PLUS S-Lang SAIL SAM76 SAS Sather Scala Scheme Scilab Script.NET Sed Self SETL Shift Script Simpol SIMPLE SIMSCRIPT Simula SISAL Slate SLIP SMALL Smalltalk SNOBOL o SPITBOL Snowball SNUSP SPARK Spice SPIN SP/k SPS (1620) Squeak Squirrel SR SSL Standard ML SBL SuperCollider Subtext Suneido SYMPL SyncCharts SystemVerilog T T TACL TACPOL TADS Tea TIE Transaction Application Language Tcl Transact-SQL teco TELCOMP Telon Tex TI-BASIC Tom TOM Topspeed tpu Trac TTCN Turbo Pascal Turing TUTOR TXL U Ubercode Unicon Uniface Unix shell Unlambda UnrealScript V Vala VBA VBScript VDM++, VDM-SL Verilog VHDL Visual Assembler Visual Basic Visual Basic.NET Visual DataFlex Visual DialogScript Visual FoxPro Visual J++ Visual Objects Vvvv VX-REXX W Water WATFIV, WATFOR WebQL Whitespace Winbatch WinDev Windows PowerShell X X++ X10 XBL xHarbour XL XOTcl XPL XPL0 XQuery XSL T Y YACC Yorick Z Z Z notation Zannon ZOP1 ZPL ZTT-zpp

Concepts behind languages

- How can we abstract over common functionality?
- How can we reason statically about our programs?
- How does a language execute a program?
- When are two programs doing the same thing?
- What does it mean for a program to be type-safe?
- Does the language maintain state?
- Does the language support garbage collection?
- What is a good (programming) language?

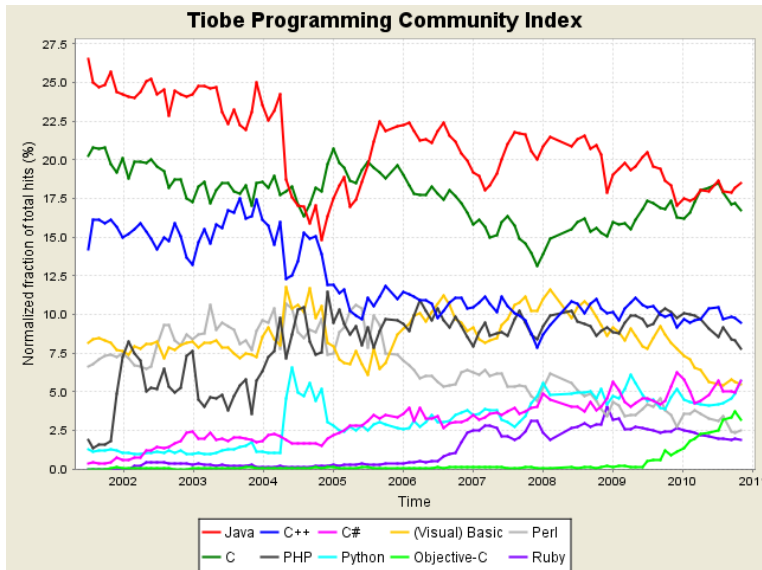
Languages evolve

Languages evolve

Programming Language	Position			
	Nov 2010	Nov 2005	Nov 1995	Nov 1985
Java	1	1	-	-
C	2	2	1	1
C++	3	3	2	8
PHP	4	4	-	-
C#	5	7	-	-
Python	6	8	10	-
(Visual) Basic	7	5	3	4
Objective-C	8	42	-	-
Perl	9	6	5	-
Ruby	10	24	-	-
Lisp	13	14	12	2
Ada	16	17	6	3

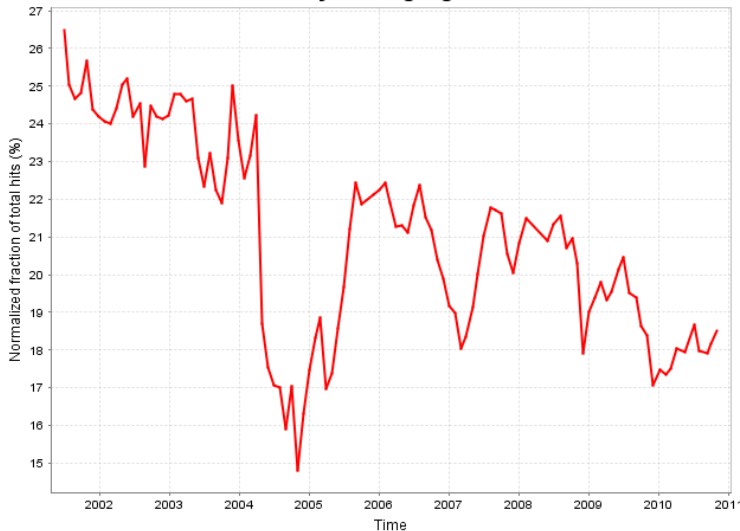
www.tiobe.com

A glimpse to the past and present



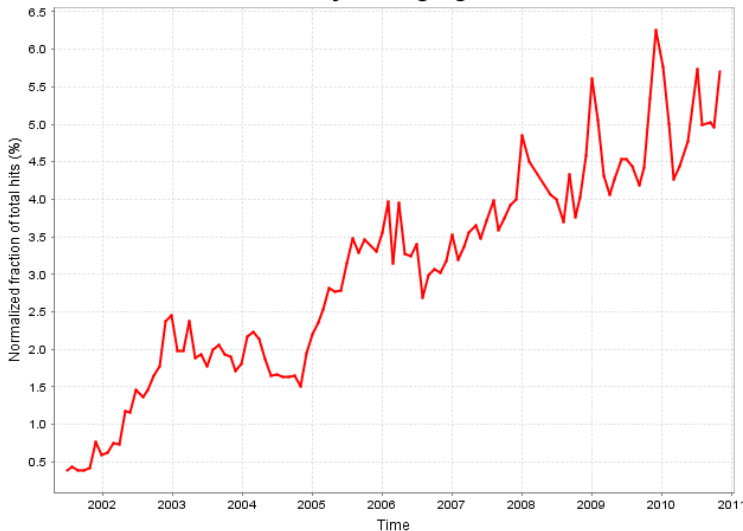
Zooming in on Java

TPCI History for language Java



Zooming in on C#

TPCI History for language C#



Take home message

- Computational thinking is a fundamental skill in many areas.
- It has fascinated human beings in the past – It continues to fascinate us today.
- Choose the right language for the problem!
- Computers (machine and human!) allow us
 - to go beyond solving problems on paper
 - to go beyond what one human being could achieve
 - to explore and understand our surrounding world
- Computer science is about computational thinking – it is challenging and tests the limits of our creativity and intelligence.

That's it!

That's it!

Thank you.