

Watch 13 minute short film before Wednesday...

The Beauty and Joy of Computing

Quest (first exam) in in 9 days!!

Lecture #4 : Creativity & Abstraction

UC Berkeley EECS Sr Lecturer SOE Dan Garcia



PREVENTING ADDICTION!

Researchers in the UK said that some gamers play up to 90 hrs a session, developing a "pathological" addiction. They say game makers need to do more, to get "sensible gaming"

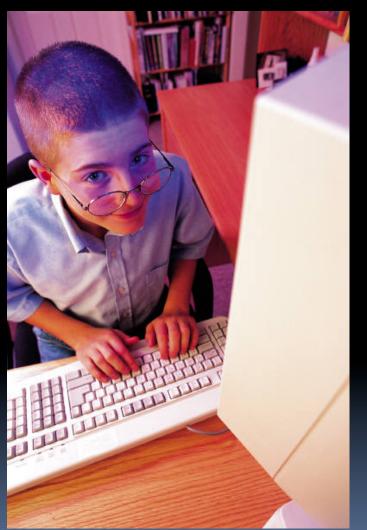
LEARN LANGUAGE FREE!

Luis von Ahn's recent project is Duolingo, which is simultaneously allowing its users to learn a second language free, while also providing a cost-effective way to get documents translated on the web.

www.duolingo.com

Computing is a Creative Activity

- "Creativity and computing are prominent forces in innovation; the innovations enabled by computing have had and will continue to have far-reaching impact.
- At the same time, computing facilitates exploration and the creation of knowledge.
- This course will emphasize these creative aspects of computing.







UC Berkeley "The Beauty and Joy of Computing" : Functions (2)

Computing enables people...

- ...to translate intention into computational artifacts.
- A computational artifact is created by human conception using software tools.
- Examples of computational artifacts include
 - digital music, videos, images
 - documents
 - combinations of these. E.g.,
 - infographics
 - presentations
 - web pages.





Garcia

UC Berkeley "The Beauty and Joy of Computing" : Functions (3)



Computing enables people...

- …to create digitally!
- Creating...
 - knowledge
 - □ tools
 - expressions of ideas
 - solutions to problems.

Creating digitally...

- requires understanding and using software tools.
- can be done by...
 - combining and modifying existing artifacts
 - creating new artifacts.







bje

Collaboration is an essential part...

- ...of creating computational artifacts.
- Collaboration facilitates multiple perspectives in developing computational artifacts.
- A computational artifact can reflect collaborative intent.









We can analyze computational artifacts...

- ...for correctness, functionality, and suitability.
- A computational artifact may have weaknesses, mistakes, or errors depending on the type of artifact.
 - For example, music created by a program may not have an error but may simply be hard to listen to.
- The functionality and suitability (or appropriateness) of a computational artifact may be related to how it is used or perceived.

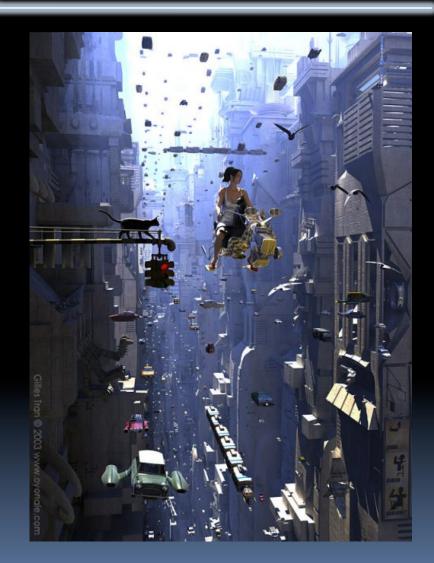






Computing extends traditional forms...

- ...of human expression and experience.
- Computer music can be created by synthesizing sounds, by sampling existing music, or by recording and manipulating sounds.
- Creating digital effects, images, and animations has impacted and transformed the movie industry.
- Computing enables creative exploration of real and synthetic phenomena.

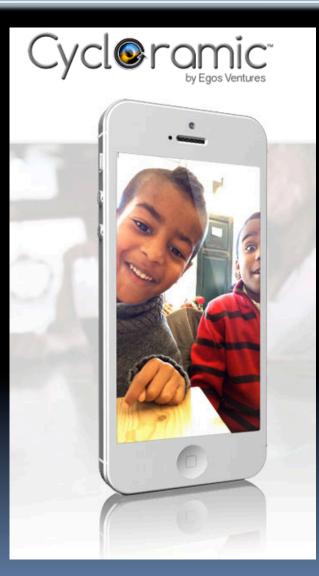






Programs can be developed...

- ...for creative expression or to satisfy personal curiosity.
- A program developed for creative expression or to satisfy personal curiosity may have visual, audible, or tactile results; or the program may affect a computer or system without such results.
- Programs developed for creative expression or to satisfy personal curiosity may be developed with different standards or methods than programs developed for widespread distribution.
- A program or the results of running a program may be shared with others.





Garcia



Programs can be developed...

- ...to solve problems, create new knowledge, or help people, organizations, or society.
 - however, the goals may be realized independently of the original purpose of the program.
- Computer programs and the results of running the programs have widespread impact on individuals, organizations, and society.











Numbers: Positional Notation

- Number Base $B \Rightarrow B$ symbols per digit:
 - Base 10 (Decimal): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
 - Base 2 (Binary): 0, 1 (In binary digits are called "bits")
 - Base 16 (Hexadecimal): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
- Number representation:
 - $d_{31}d_{30} \dots d_1d_0$ is a 32 digit number
 - value = $d_{31} \times B^{31} + d_{30} \times B^{30} + ... + d_1 \times B^1 + d_0 \times B^0$
- Binary Obl1010 = $1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$ = 16 + 8 + 2= 26
- Hex $0x1A = 1 \times 16^{1} + 10 \times 16^{0}$ = 16 + 10 = 26
- One hex digit (four bits) is a "nibble". Two (eight bits) is a "byte" (values 0-255)
- N bits ⇔ at most 2^N things







Abstraction (revisited): Numbers

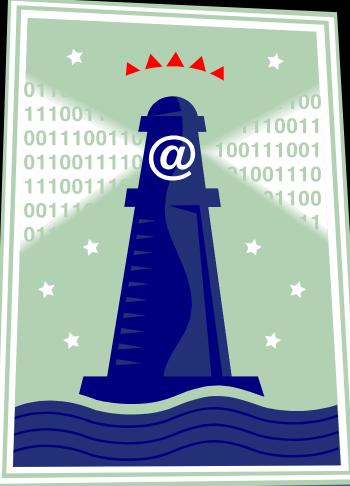
- Number bases, including binary and decimal, are used for reasoning about digital data.
- Bits represent binary data using base two digits: zero and one.
- Hexadecimal, or base-16, is often used in reasoning about data such as colors in images.
- Different bases help in reasoning about digital data; digital data is stored in bits.





Abstraction (revisited): Digital Data

- A combination of abstractions is used to represent digital data.
- At the lowest level all digital data are represented by bits.
 - Said another way, bits can represent anything!
- Bits are grouped to represent higherlevel abstractions including numbers and characters.
 - Logical values? 0 \rightarrow False, 1 \rightarrow True
 - Colors? 00 \rightarrow Red, 01 \rightarrow Green, 10 \rightarrow Blue
 - Characters? $00000 \Rightarrow 'a', 00001 \Rightarrow 'b', \dots$
- Higher-level abstractions such as Internet protocol (IP) packets, images, and audio files are comprised of groups of bits that represent different parts of the abstractions.





Binary Sequences to Represent Data

- A finite representation is used to model the infinite mathematical concept of a number.
- In many programming languages the fixed number of bits used to represent integers limits the range of integer values, and mathematical operations can result in overflow or other errors.
- In many programming languages the fixed number of bits used to represent real numbers (represented as "floating-point numbers") limits their range, and mathematical operations can result in round-off and other errors.





June 5, 1933, in Toronto, Ontario,

B.Sc. 1954; M.Sc. 1956; Ph.D. 1958 (all in

Mathematics University of Toronto)

University of Waterloo, Canada, 1998;

Chalmers Inst., Goteborg, Sweden, 1993

Honorary Doctor of Mathematics,

Honorary Doctor of Mathematics

Postdoctoral fellow, Cambridge University Mathematical Laboratory

(UK) 1958-1960; Assistant/Associate Professor of Mathematics, University of

Mathematics/Electrical Engineering &

Computer Science (currently Emeritus

sor), University of California

First ACM George. E. Forsythe Memorial

1989; ACM Fellow (1994); SIAM John von Neumann Lecture (1997); IEEE Emanuel R. Piore Award (2000);

Foreign (Canadian) Associate, National

Academy of Engineering (2005).

Award (1972); ACM Turing Award,

Toronto 1960-1968; Professor of

Berkeley 1968.

Canada

EDUCATION

WILLIAM ("VELVEL") MORTON KAHAN United States – 1989

CITATION For his fundamental contributions to

For his fundamental contributions to numerical analysis. One of the foremost experts on floating-point computations. Kahan has dedicated himself to "making the world safe for numerical computations"!

SHORT ANNOTATED	ACM DL	RESEARCH	ADDITIONAL	
BIBLIOGRAPHY	AUTHOR PROFILE	SUBJECTS	MATERIALS	

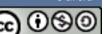
William Kahan was born in Canada in 1933 and grew up around Toronto in a family of Jewish immigrants. Its mother created a successful dress design business and his father ran a factory. He was known within his family as Velvel, or "little woll," a name still favored by his friends over the Anglicized "William". As a young man Kahan loved to fix mechanical and electronic devices, an inclination tostered by a summer job repairing surplus wartime equipment. He related this love of tinkering throughout his little. He kept a pair of customized 1984 Peugeot 505 running for more than twenty years, and for a long time he eschewed laser printers in favor of a homebre wystem optimized for printing equations on a dot matrix printer.

Kahan learned to program during the summer of 1953 as an undergraduate majoring in mathematics at the University of Toronto. The university was a center of early computer development and use, and owned FERUT, one of the world's first commercially manufactured computers -the second Manchester Ferranti Mark I ever built. Remaining at Toronto as a graduate student, he focused his studies on numerical analysis and explored the new possibilities in applied mathematics that were opende up by the use of computers. He made extended visits to two other computing centers: he spent the summer of 1957 working on the ILLIAC I at the University of Illinois, and after completing his Ph.D. in 1958 spent two years with the EDSAC team at Cambridge University.

He returned to Toronto in 1960 as a faculty member, where his research focused on the error analysis of numerical computations. For Kahan this involved not just determination of the accuracy of calculated results, but also the design of new software and architectural features to improve accuracy while maintaining high performance. He created an integrated system of mathematical routines, compiler tweaks and operating system modifications for the university is IBM 7044 computer to help porgrammers create accurate, high performance floating point code. Kahan played a leading role in the numerical analysis subgroup of the IBM computer user group SHARE; and spantheaded its successful campaign in 1968-1967 to force IBM to fix design flaws in the arithmetic of its new System x380 computers.

Kahan left Toronto in 1968 for the University of California, Berkeley, to botster its newly created department of computer science. (Another Turing Award winner, Richard M. Karp, joined the same year as Kahan). Kahan credits the move to a combination of the attractions of Northern California for his children, who had been struck by its beauty during his recent sabbatical at Starford University, and his dwindling faith in the future of high performance computing in Canada following the colapse of the country's high technology defenses sector.



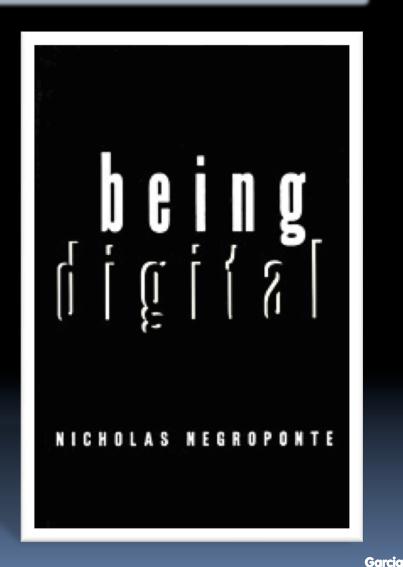


Garcia

UC Berkeley "The Beauty and Joy of Computing" : Functions (13)

Interpretation of a Binary Sequence...

- ...depends on how it is used (e.g., as instruction, number, text, sound, or image).
- The sequence of bits that represents...
 - ...an instruction may also represent data processed by that instruction.
 - ...a character/letter may also represent a number.
 - ...a color in an image may also represent a sound in an audio file.

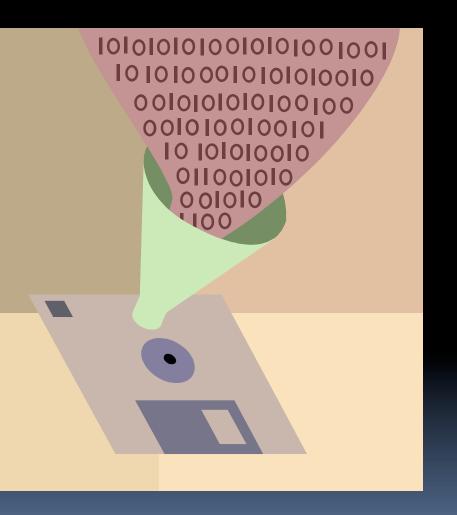






SW and HW built on multiple abstractions!

- Software is built using low- and high-level abstractions...
 - such as expressions, statements, data types, functions, and libraries.
 - that represent hardware, such as device drivers and game controllers.
- Hardware is built using low- and high-level abstractions such as chips, memory, and storage.







Binary Data is processed by...

- ...physical layers of computing hardware, including gates, chips, and components.
- A logic gate is a hardware abstraction that models a Boolean function. (and
- A chip is an abstraction composed of low-level components and circuits that performs a specific function such as memory, CPU, encryption, and more.
- A hardware component can be low level like a transistor or high level like a video card.

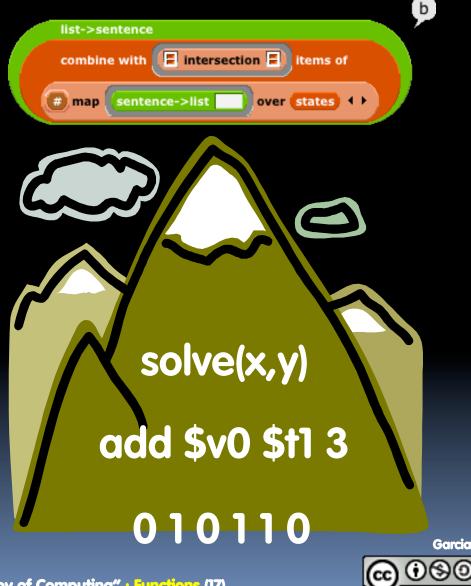






Programming languages, from low to high level...

- ...are used in developing software.
- Low-level programming languages, such as assembly, are closer to the machine level and provide fewer abstractions for the programmer.
- High-level programming languages provide more abstractions for the programmer and are easier for humans to use for reading and writing code.
- Code in a high-level programming language is typically automatically translated into code in a lowerlevel language to be executed on a computer; this is done by a compiler or an interpreter.

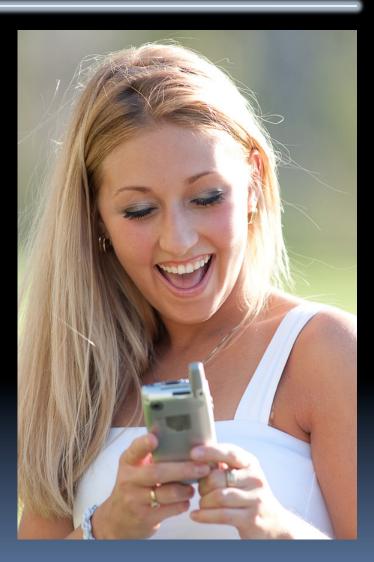




UC Berkeley "The Beauty and Joy of Computing" : Functions (17)

Abstractions everywhere!

- Applications and systems are designed, developed, and analyzed using levels of hardware, software, and conceptual abstractions.
 - E.g., Mobile applications and systems
 - E.g,. Web services (both an application and a system)









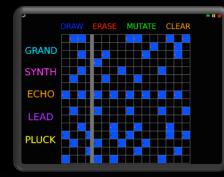
Creativity

 You will create interesting and relevant artifacts with the tools and techniques of computer science.

Abstraction

- This course will include examples of abstractions used in modeling the world, managing complexity, and communicating with people as well as with machines.
- You will learn to work with multiple levels of abstraction while engaging with computational problems and systems.









UC Berkeley "The Beauty and Joy of Computing" : Functions (19)

Garcia