Computer Science: The Big Picture

- Computer science is a lot of things to a lot of people — some accurate, some misleading

- Harel notes that *computer science* as a term for its field is akin to calling surgery "knife science"

  - ◇ It isn't really *wrong*

  - ◇ But it doesn't capture the essence of the field, the way *mathematics, physics, music, history*, or *accounting* do (among many others)

The Core: Algorithms

- Regardless of the term's quality, we're stuck with it: it's called *computer science*

- Nevertheless, there is an overall consensus on the soul of the field, and that is the *study of algorithms*

- An *algorithm* is a set of steps that defines how a task is performed — note how the word "computer" is not to be found in that definition

- Alongside algorithms are *abstractions*: the separation of an idea's external meaning from its internal details

Algorithms are Everywhere

Navigation	How to drive from LMU to Las Vegas How to reach the North Pole
Cooking	How to make chocolate mousse How to poach an egg
Instruction Manuals	How to use the Hole-Hawg Drill How to play Monopoly
Financial Transactions	How to balance your checkbook How to make change How to calculate your monthly loan payment
Medical Procedures	How to perform an appendectomy How to interpret an MRI scan How to diagnose bubonic plague
Dance Steps and Routines	How to dance the six-count swing How to accomplish the salsa cross-body lead How to dip your dance partner
Daily Activities	How to replace a light bulb How to fix a leaky faucet

- If you've ever wondered about…

  ◇ What properties or characteristics these processes or procedures share

  ◇ Comparing them in terms of complexity, degree of difficulty, or speed

  ◇ How they can be "encoded" or communicated from one person to another

  ◇ Whether they have limitations or whether they are even possible — for example, is there an algorithm for "how to find your true love?"

  …then computer science is the place to be! It's time to change your major or go to graduate school   ;–)

Abstractions Go
Hand-in-Hand

- Note abstractions are inextricable from the algorithms mentioned previously:
  - ◇ The very notion of "driving" is an abstraction — it includes how to use the gas pedal, how to shift gears, how to brake, etc.
  - ◇ In cooking, certain activities eventually get abstracted out: for example, a recipe for eggs benedict will simply ask you to poach an egg, but may not tell you how to to that in detail
  - ◇ Dance moves frequently build upon each other

- Abstractions pertain to both the activities within an algorithm (e.g., driving, poaching an egg), and to the objects (things, nouns) in that algorithm
  - ◇ "Las Vegas" and "the North Pole" are abstractions for specific locations in the world — simpler to use than latitude and longitude, for example
  - ◇ The Hole-Hawg Drill consists of other components — switches, drill bit, power supply — that we may or may not use directly
  - ◇ The concept of a "loan" encapsulates details such as borrowing money, paying interest, etc.
- One wonders what the world might be like if *everyone* learned a little computer science to supplement their own fields, careers, or specialties

A Little History

- So you see, algorithms and abstractions have been around us forever

- Euclid's procedure for finding the greatest common divisor (GCD) of two positive integers is widely considered to be the first non-trivial algorithm

- The word *algorithm* itself is derived from the Persian mathematician Mohammed al-Khowârizmî, who wrote a manual for how to add, subtract, multiply, and divide decimal numbers — "al-Khowârizmî" translates to "Algorismus" in Latin

Eventually we became aware of algorithms as concepts (abstractions!) in their own right, and began to study them explicitly — not surprisingly, mathematicians led the initial charge:

- Kurt Gödel's (1906–1978) incompleteness theorem(s) show that certain statements in a formal system cannot be proven formally within that system

- Alan Turing's (1912–1954) work allows us to reason formally about algorithms — in other words, it allows us to prove (or disprove) statements about algorithms

  ◇ Key ideas: the *computability* and *decidability* of certain problems — it has been proven that *noncomputable* or *undecidable* problems do exist

- Other key contributors include Alonzo Church, Andreĭ Markov, John von Neumann, Stephen Kleene, and more

Computing Machines (Finally!)

- Meanwhile, as far back as Euclid and al-Khowârizmî, devices that facilitated or performed algorithms have been designed and sometimes built:
  - ◇ Abacus
  - ◇ Slide rule
  - ◇ Early computing machines by Pascal (1623–1662) and Leibniz (1646–1716)
  - ◇ Jacquard's (1752–1834) weaving loom (ca. 1801)
  - ◇ Hollerith's (1860–1929) punch cards (1890)

- Special mention: Charles Babbage (1792–1871)

  - ◇ Built a demonstration *difference engine* that can compute certain values through a technique called *successive differences*

  - ◇ Specified an *analytical engine* that can do far more than the difference engine, as specified by holes punched into cards (inspired by Jacquard's weaving loom)

- The analytical engine was never built, but it *was* programmed — Augusta Ada Byron, Countess of Lovelace (1815–1852), published examples of how the analytical engine can be instructed to do a variety of tasks…because of this, she is considered to be the world's first-ever *programmer*

Mechanics to Electronics

- The previous devices where all mechanical — they worked through gears, cogs, levers, etc.

- The 20th century brought about *electronic* machines, driven by technologies that progressed from mechanical relays, to vacuum tubes, to transistors, to integrated circuits — and beyond

- Notable machines: Stibitz's machines (1930s), Aiken's Mark I (1944), the Atanasoff-Berry machine (1937–1941), Zuse's Z3 (1941), Flowers's Colossus (1944), Mauchly and Eckert's ENIAC (1946), and many more

"Computer" Meets "Science"

- While the *soul* of computer science may be the study of algorithms, most of its *body* involves getting *machines* to perform these algorithms for us

- Thus, it is *Alan Turing* who is viewed as the father of computer science, because his work bridges the gap between the pure concept of algorithms and the tangible reality of computing machines

- His *Turing machine* is the theoretical foundation behind the very real machines and devices that surround us today — this is the *science* behind the *computer*

Epilogue: Computers, Ethics, and Society

This wouldn't be LMU if we stopped with the pure science and technology behind computer science — many aspects of society have been fundamentally transformed by the field's milestones and achievements

- How do we protect intellectual property in the age of instant access and rapid duplication/dissemination?
- Does advancing technology stratify or unite society (i.e., "the digital divide" vs. the "wired" world)?
- Does (should) the baseline of education or literacy change as technology advances?

And let's not forget how technology may (or may not) affect us as human beings and social creatures

- How does instantaneous and omnipresent communication change social customs and the way we interact with others?
- How does one balance increased automation and the possible obsolescence of certain skills or jobs?
- Is access to certain technologies a right or a privilege? How does this affect our sense of what it means to be a human being?

Computers and technology — and, by extension, computer science — have a unique and radical effect on society…as we get more technical in this course, we must strive not to forget about any *non*-technical consequences