CS17700

Hello world

Instructors

- Voicu Popescu, instructor
  - popescu@purdue.edu
  - http://www.cs.purdue.edu/homes/popescu/
- Lorenzo Martino, instructional coordinator
  - lmartino@purdue.edu
- Teaching assistants

2

CS17700

- Two major goals
  - An introduction to Computer Science principles
  - An introduction to Computer Science practice using Python
- Targeted audience characteristics and needs
  - No Computer Science background
  - Collaboration with computer scientists
  - Use of complex computer science tools (i.e. software)
  - Development of custom computer science tools

3

Course organization

- Webpage
  - http://wiki.cs.purdue.edu/177
- Lecture
  - New concepts are introduced
- Recitation
  - Concepts are explained in more detail, reinforced
- Lab
  - Concepts are practiced

4

Communication

- Piazza ("marketplace" in Italian)
  - Online forum where students post questions and students and instructors post answers
  - Better scalability then direct, one to one email
  - Instructions posted on class webpage
  - Policies
    - Do not post lab or project solutions, partial solutions, incorrect solutions (cheating)
    - Use #private tag if not sure
    - Make questions general, clear, and concise

5

Communication

- Piazza
- Office hours
  - See webpage for details
  - Not a substitute for recitation or labs
- Instructor available after class for questions
  - I'll stay as long as needed (hallway if need be)
- In class via iClicker

6

Syllabus overview

- Computer Science Principles: ~6 weeks
  - Data, data structures, introduction to algorithms, basic algorithms, recursion
- Computer Science Practice: ~6 weeks
  - Programming in Python (data structure implementation, control flow, functions, debugging, recursion, advanced data processing)
- Computing and society: ~3 weeks
  - Internet, cyber security, and societal impact of computing

Resources

- Slides
- Text book
  - Python Programming: An Introduction to Computer Science. John Zelle. Second Edition. Franklin, Beedle & Associates Inc.
- Wiki Book (online book)
  - http://en.wikibooks.org/wiki/Python_Programming

Grading

- Attendance of lectures and recitations
  - 5% of course credit
  - No attendance taken the first week
  - After 4 lectures missed, 1% off for every additional lecture absence
  - After 2 recitations missed, 1% off for every additional recitation absence
  - This includes all absences (e.g. interviews, conferences, short term health issues, work, etc.)

9

Grading

- Attendance of lectures and recitations
  - 5% of course credit
- Weekly lab
  - 25% of course credit
- Projects
  - 5 x 5% = 25% of course credit
  - Late policy
    - <24h -20% of project credit
    - >24h & <48h -50% of project credit
    - >48h -100% (no credit)
- Midterm examinations
  - 2 x 12.5% = 25% of course credit
- Final examination
  - 20% of course credit

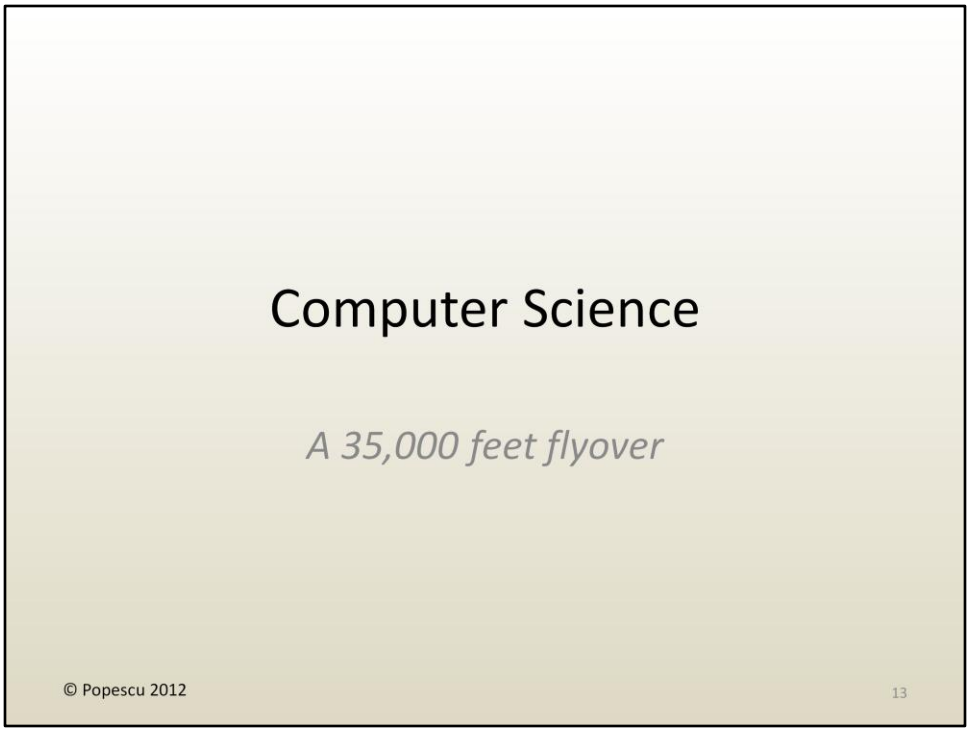
Policies

- All CS 17700 students have to
  - Familiarize themselves with CS policies
    - http://spaf.cerias.purdue.edu/cpolicy.html
  - Confirm knowledge of and adherence to CS policies
    - http://www.cs.purdue.edu/
    - Log into CS Portal using Purdue Career credentials
    - Click on “Academic Integrity Policy” on the left tab
    - Read policies carefully
    - Logging in is equivalent to e-signature

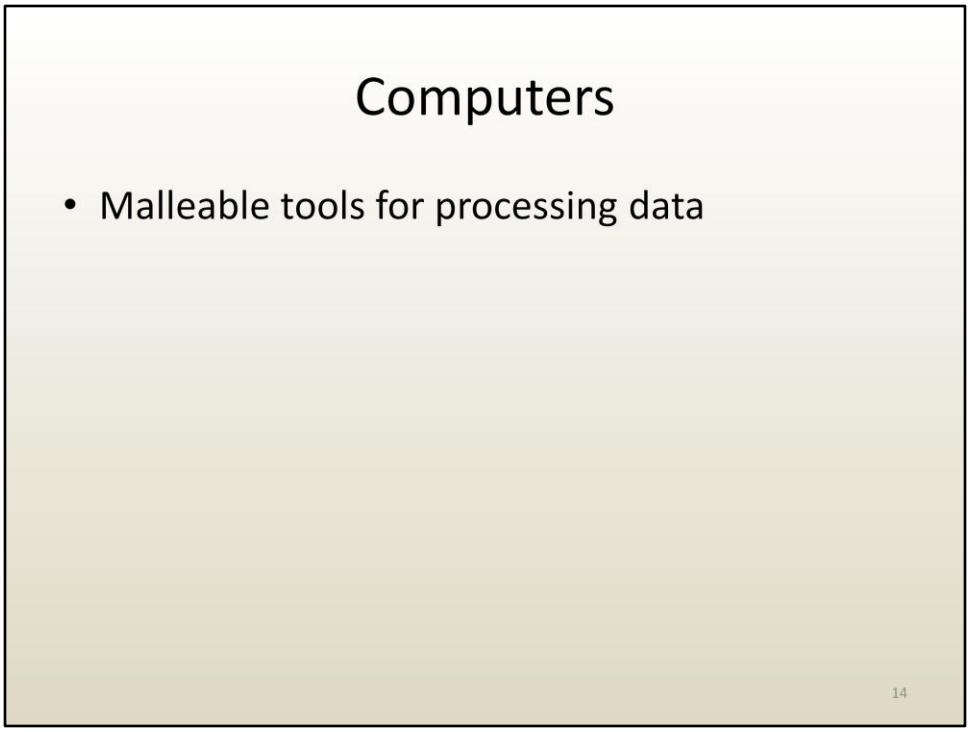
11

CoS Teaming Requirement

- SCI 210
  - Principles of working in teams
  - Blackboard module, first 6 weeks of the semester
- Two or three CS 17700 team projects
  - Practice of working in teams
  - Project questions will evaluate understanding of teaming



Here is a very high level and very brief overview of computer science. We will spend the rest of the semester adding details to this overview.



Let’s think about what computers are. Computers can be described as malleable tools for processing data. For this to make more sense, let’s first see what data is, why and how data is processed, and how computers process data. We will get back to this slide at the end of the lecture.

Data

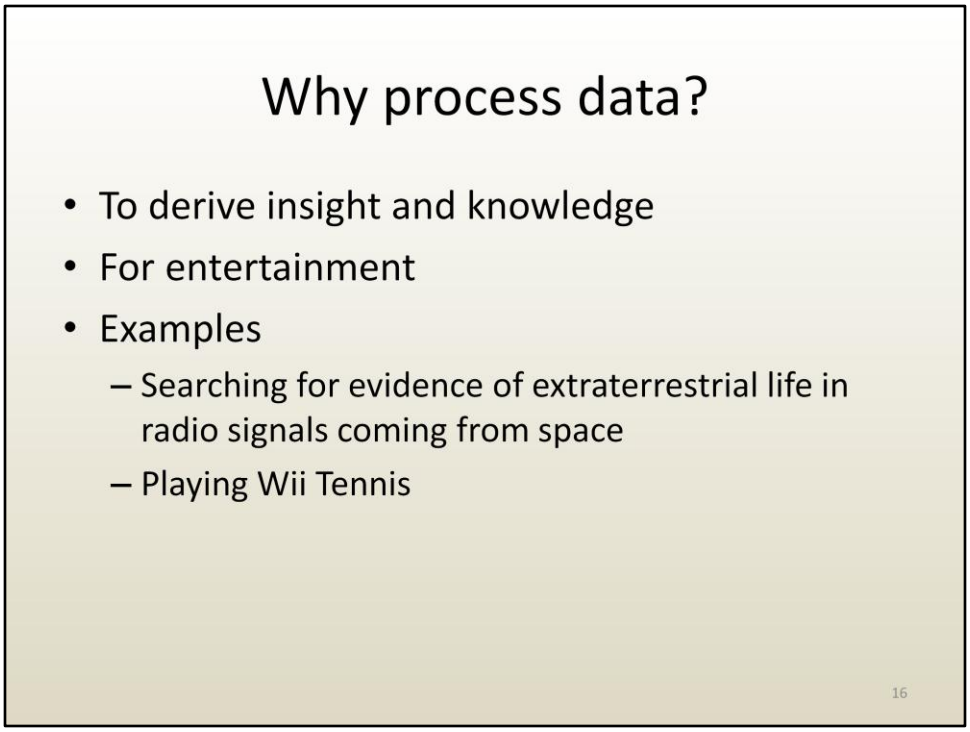
- "Factual information used as a basis for reasoning, discussion, or calculation" M. Webster
- Can be stored, transformed, and transmitted
- Examples
  - Names of people in this class
  - A self-portrait by Van Gogh
  - Results of a molecular dynamics simulation

15

What is data? The Webster dictionary defines it as "factual information used as a basis for reasoning, discussion, or calculation".

It is also the case that data can be:
- Stored, such that it is available in the future
- Transformed, in other words new data is obtained by processing input data
- Transmitted, it can be communicated over great geographic distances.

Here are some examples of data.

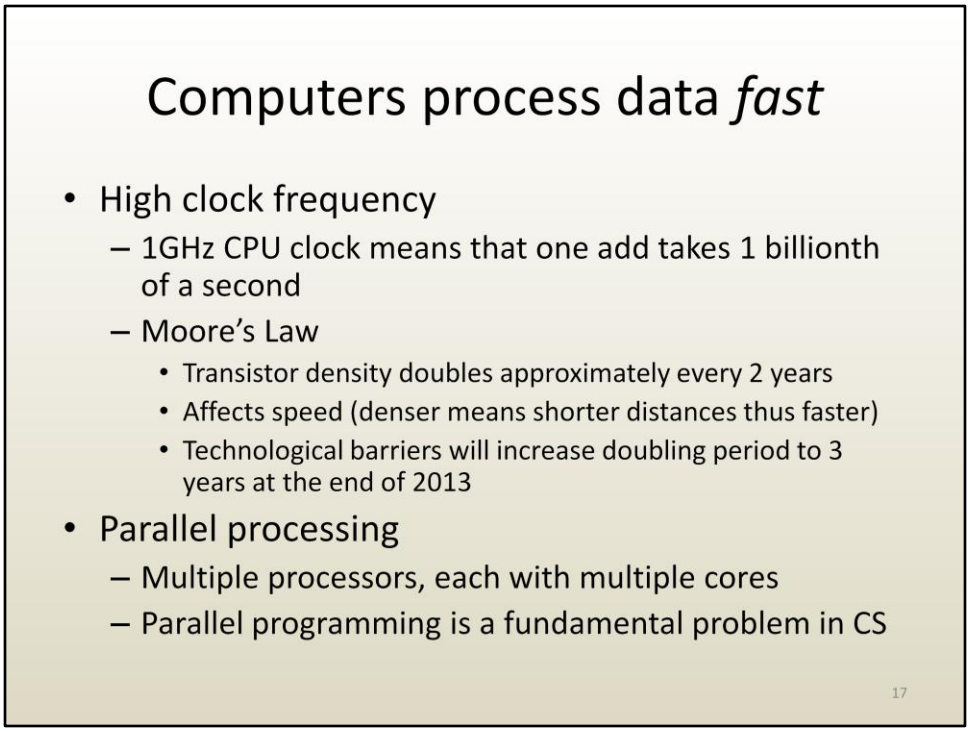


What is the motivation behind processing data?

Processing data can lead to insight and knowledge. It is processing that transforms the observed or simulated data into knowledge.

For example one has to process the radio signals coming from space in order to detect patterns that might indicate extraterrestrial communication.

As you surely know, computers also process data for entertainment purposes.



Data processing by computers has several important characteristics.

First, computers process data very quickly.

One reason for this is that the computer clock frequency is very high. Think of the clock frequency as of the speed of the assembly line.

Computers have Central Processing Units with clocks in the Gigahertz's. 1 GHz means 1 billion ticks per second. Since one add takes one tick, that means one billion additions per second. CPU speeds have been increasing as chip transistor density increased. Moore noticed that transistor density doubles about every 2 years—that is called Moore's law. However, transistor density increases have recently slowed down. It is estimated that the doubling period will be 3 years by the end of 2013.

A second important reason why computers process data quickly, is that data is processed in parallel. Multiple pieces of data are processed simultaneously using multiple processing units. As you know, over the last years, as CPU speed has stopped increasing, computer manufacturers now make computers with multiple processors, each with multiple cores. This continues to increase the raw processing power of computers. However, it is not easy to take advantage of the tens of cores available. Data processing has to be distributed over the many cores. Parallel programming is a fundamental problem in computer science.

Computers process data *accurately*

- Computer HW is accurate
  - No arithmetic errors
    - Almost none (Pentium FDIV bug caused division errors)
  - No memory or disk reading errors
    - Unless hardware failure

66MHz Intel Pentium with the FDIV bug

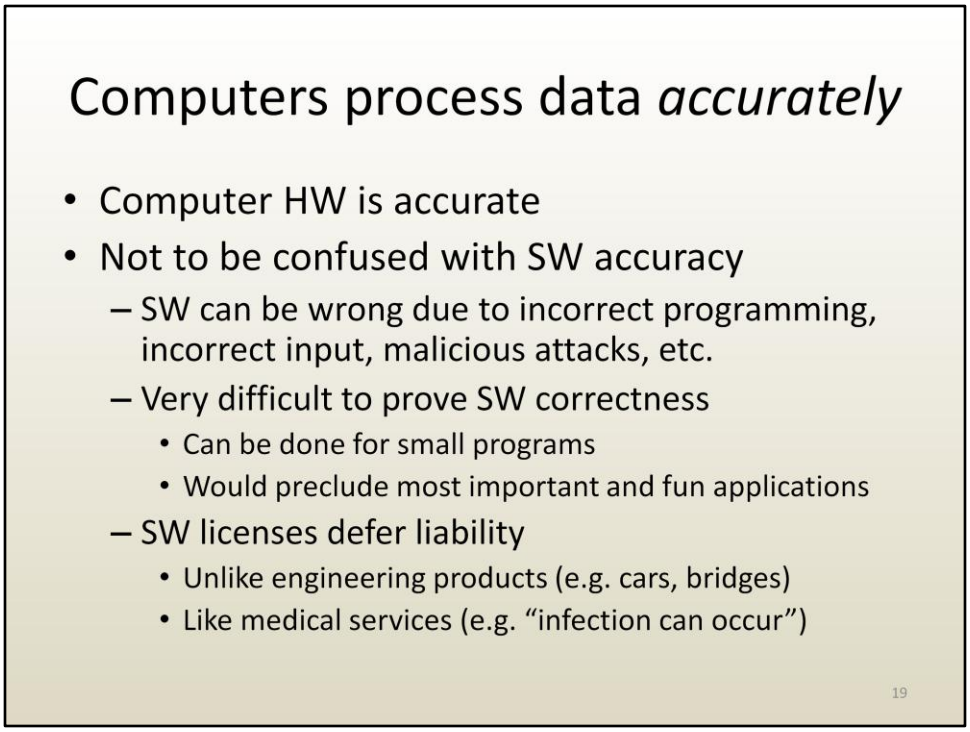
18

We have seen that data processing by computers is fast.

It is also accurate. What we mean, is that computing hardware is accurate.

It is unlikely that your calculator will make an arithmetic error. Do know though that ten years ago or so, Intel did put out a chip that, in some special cases, was not computing divisions correctly.

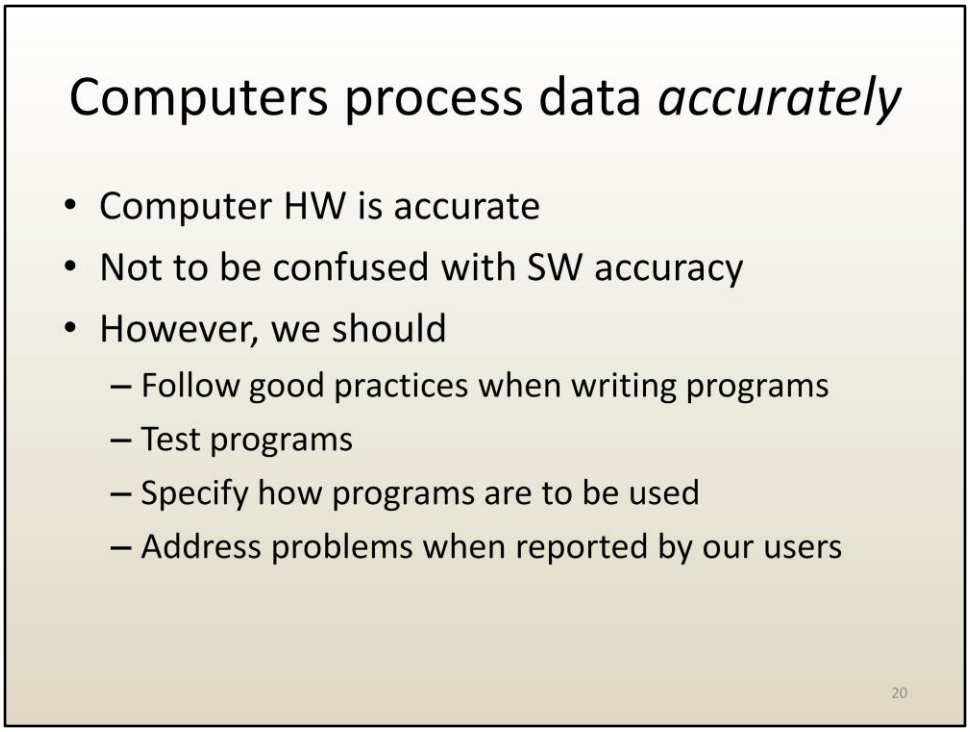
Data is written and read correctly from memory or drives, in the absence of hardware failure.



Whereas computing hardware processes data accurately, this does not mean that all software will process data as the application user expects.

Software can be wrong
- due to incorrect programming, or programming bugs,
- due to providing invalid input, which the software rightly or wrongly does not expect
- due to viruses, malware and other malicious attacks.

It is not feasible to prove that large pieces of software are mathematically correct. Whereas one can prove small programs as being mathematically correct, eliminating any possibility for a bug, requiring correctness proofs for all software is impractical. Software companies wouldn't be able to do it, and most important and exciting applications running on computers would not exist. Software licenses typically defer liability: "use at your own risk".



However, we should do our best for avoiding software errors. This includes following a set of rules when writing programs, testing programs thoroughly on relevant data, informing users how the program should be used, and addressing problems reported by the users.

Computers process data *accurately*

- Computer HW is accurate
- Not to be confused with SW accuracy
- However, we should
  - Follow good practices when writing programs
  - Test programs
  - Specify how programs are to be used
  - Address problems when reported by our users
  - "Program correctness is not possible nor required, and Microsoft, Adobe, and Apple can't do it either" defense will not fly in CS 17700

21

Of course, you cannot defend a bad code in CS 17700 by pointing to bugs in commercial software, or by pointing out that it is difficult to prove program correctness.

Computers excel at *low-level* data processing

- Computers can easily
  - Search through billions of words to find a given word
  - Increase the brightness in billions of images
  - Sort billions of health records alphabetically
- Computers have a harder time
  - Understanding natural language (e.g. humor, irony, sarcasm)
  - Deciding which of two paintings is better
  - Reconstructing the 3-D geometry of a real world scene from photographs
  - Not impossible, subject of ongoing research

22

So far we have argued that computers process data quickly and accurately.

It is also true that computers excel at low-level data processing, and that computers find high-level data processing tasks more challenging.

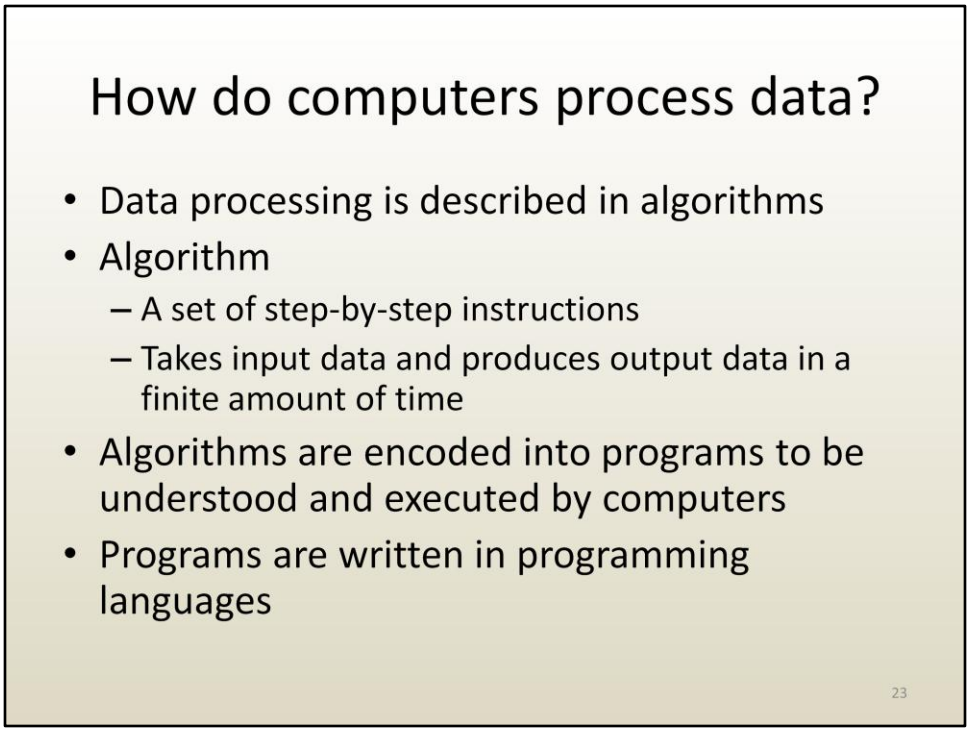
Computers can crunch numbers faster and more accurately than any human. But computers have a difficult time with tasks that are quite simple for humans.

For example computers have difficulty understanding natural language.

Computers have a difficult time quantifying aesthetics.

Also whereas we have no problem estimating the geometry of the real world from 2 images, computer reconstruction of 3-D geometry from image remains challenging.

Please note, none of these problems are impossible to solve using computers. To the contrary, computer science research is making great progress towards solving them. However, the point that computers are intrinsically great at crunching numbers remains.

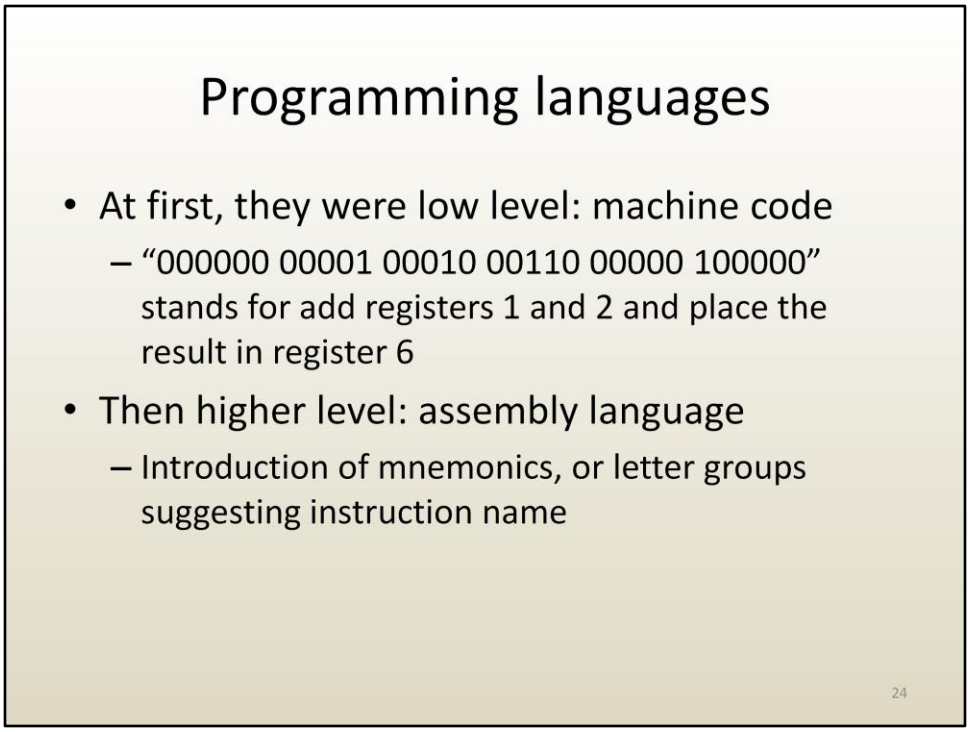


We have seen that computers process data quickly, accurately. But how exactly do they do that?

The processing needed for an application is described in an algorithm.

An algorithm is a set of step by step instructions for processing data. The algorithm takes input data and produces output data in a finite amount of time. In other words, the algorithm has to finish.

In order for a computer to understand and execute an algorithm, algorithms have to be encoded into programs. Programs describe algorithms in special languages, called programming languages.



At the beginning of computing, that is over 50 years ago, programming languages were very low level. Even calling them a language would be a stretch. They were more like a numerical code, or machine code, a sequence of 0's and 1's. Programming computers directly in machine code was very tedious.

To simplify the programming task, computer scientists developed assembly languages, which used letters in addition to numbers. Groups of letters defined mnemonics which suggested the name of the instruction.

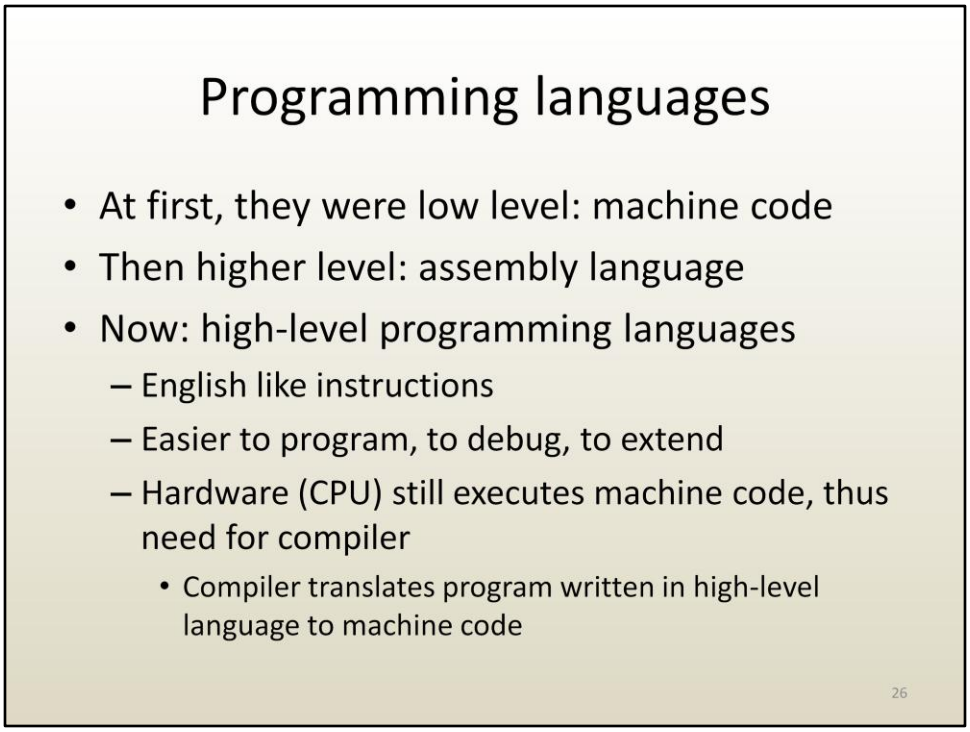
Motorola MC6800 Assembly Language

```
                ************************************
                * FUNCTION: INCH - Input character
                * INPUT: none
                * OUTPUT: char in acc A
                * DESTROYS: acc A
                * CALLS: none
                * DESCRIPTION: Gets 1 character from terminal

C010 B6 80 04   INCH    LDA A  ACIA      GET STATUS
C013 47                 ASR A            SHIFT RDRF FLAG INTO CARRY
C014 24 FA              BCC    INCH      RECIEVE NOT READY
C016 B6 80 05           LDA A  ACIA+1    GET CHAR
C019 84 7F              AND A  #$7F      MASK PARITY
C01B 7E C0 79           JMP    OUTCH     ECHO & RTS
```

25

In this Motorola assembly language example, JMP most likely stands for jump, LDA for load, and AND for the logical operation “and”. Assembly language is a little bit easier to read and write than machine code, but still quite far from the way humans communicate. In the early days, programmers had to adapt to computers, they had to learn the low level computer language.



Nowadays, most programmers never have to look at code in assembly language. Computer science now uses high-level programming languages, which are quite human friendly. They resemble the English language. Programmers have a much easier time programming, finding bugs, and extending programs because of the high-level languages in which programs are written.

The high level language is not executed directly by hardware, it still has to be translated into machine code, but there are software tools for doing that automatically. In other words, there are programs that take a program written in a high-level language and produce the corresponding code. Such a program is called a compiler.

High-level programming language example

```
sum = 0;
for(i = 0; i < 10; i++)
        if (a[i] > 0)
                sum = sum + a[i];
```

27

Here is an example of a program written in a high-level language. It has some English words in it: for, if.

What does it do? One hint: the equal sign means assign the quantity to the right to the variable (i.e. container, piece of memory) to the left.

First we set a variable sum equal to 0. Then, for all 10 elements of a vector a (i.e. sequence, array), if the element with index i is greater than 0, add it to the variable sum. At the end, sum will store the sum of all positive numbers in the array.

High-level programming language example

```
// this program computes the sum of the
//       positive numbers in an array of 10 numbers
sum = 0; // initialize the sum to 0
for(i = 0; i < 10; i++) // traverse the array, starting from
                        // first number, until the last, one at
                        // the time
        if (a[i] > 0) // if the current number is positive
                sum = sum + a[i]; // add it to the sum
```

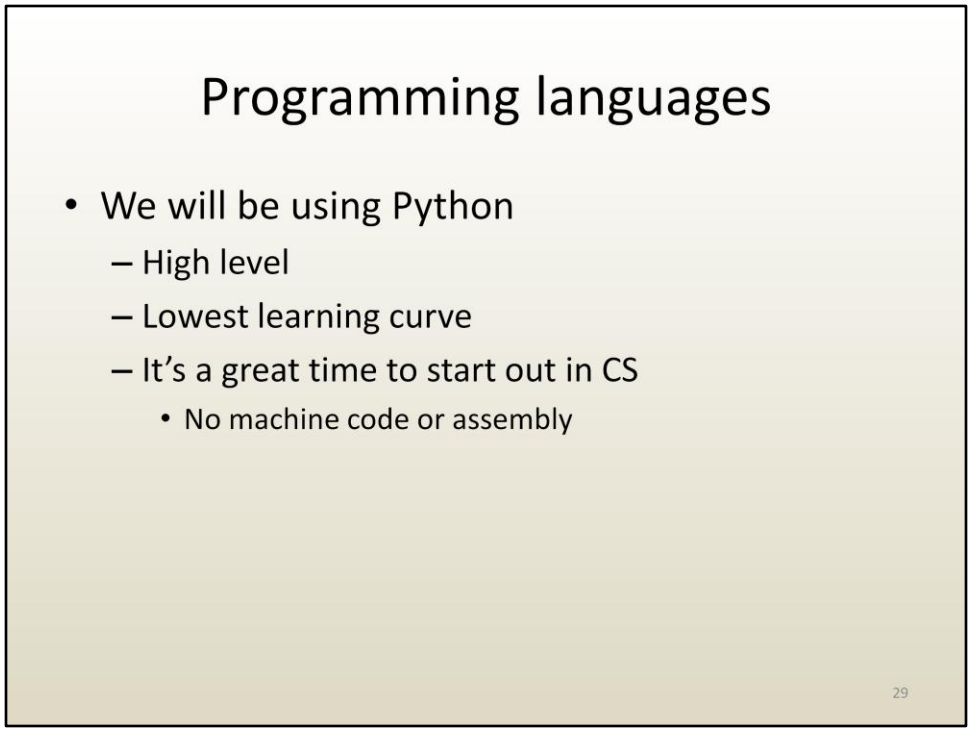
28

One more time.

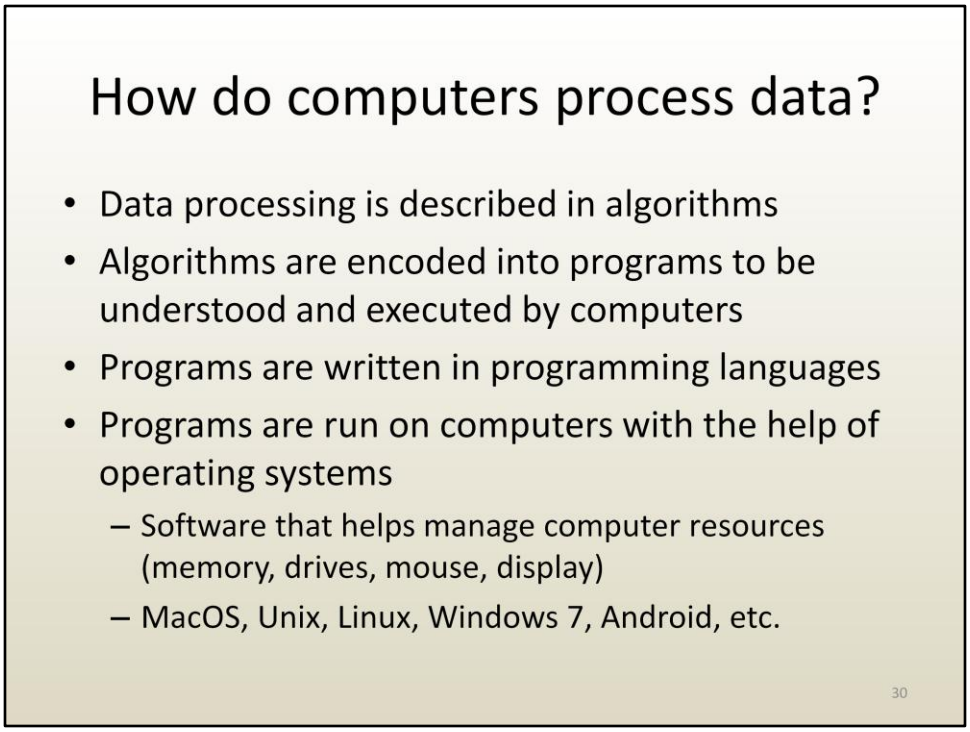
Initialize variable sum to 0.

Then do some processing repeatedly. Start i at 0. Keep repeating the process while i is less than 10. From one iteration to the next, increment i by 1. Here "++" is a fancy way of writing i = i + 1.

What is the processing that should be repeated? This process: see if a_i is positive, and if yes, add it to sum.



In this class we will be using Python, which, is high level. Python has a low learning curve. You do not have to worry about assembly languages or machine code. By the way, the example on the previous slide is in C, one of the most popular programming languages today.



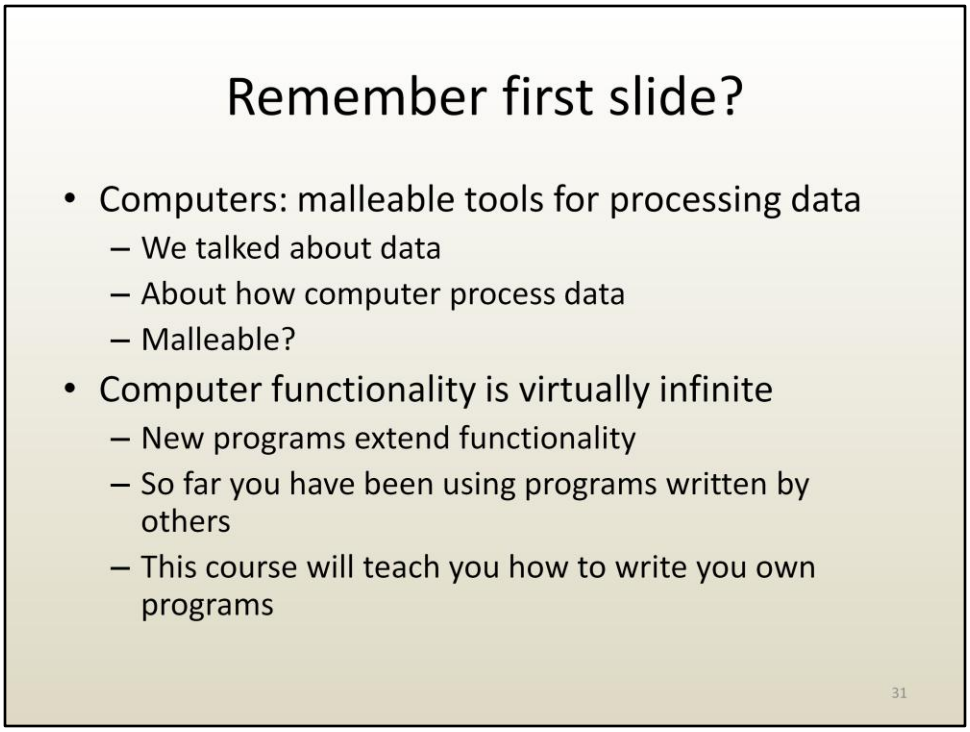
Let's recap. How do computers process data?

Data processing is described in algorithms. At first, algorithms can be specified in a variety of ways. For example, as a sequence of step by step instructions in English.

Then algorithms are encoded into programs to be understood and executed by computers. A program is a formal description of an algorithm.

Then special software, called a compiler, takes the program and translates it into machine code. This step is automatic, carried out by computers.

Then special software, called an operating system, runs the machine code to actually execute the desired data processing.



Let's go back to our first slide. We called computers "malleable tools for processing data". We have by now justified the "tools for processing data" part. But how about malleable?

Computers are tools that can be taught how to solve any number of problems. They are not rigid tools intended to solve a single problem. Computer functionality is extended by writing new programs. So far you have been using programs written by others. If you wanted to take advantage of the malleability of computers, you would buy additional software. Buy tax software and now your computer also knows how to do your taxes.

In this course you will learn how to write your own programs.