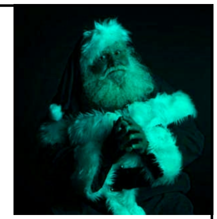
Introduction to Computing

CS A109

Course Objectives

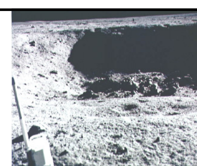
- Goal is to teach computation in terms relevant to non-CS majors
- Students will be able to read, understand, modify, and assemble from pieces programs that achieve useful communication tasks: Image manipulation, sound synthesis and editing, text (e.g., HTML) creation and manipulation, and digital video effects.
  - We will give you examples to use as a basis when you write your own programs
- Students will learn what computer science is about, especially data representations, algorithms, encodings, forms of programming.
- Students will learn useful computing skills, including graphing and database concepts

```
def clearRed(picture):
  for pixel in getPixels(picture):
    setRed(pixel,0)
```

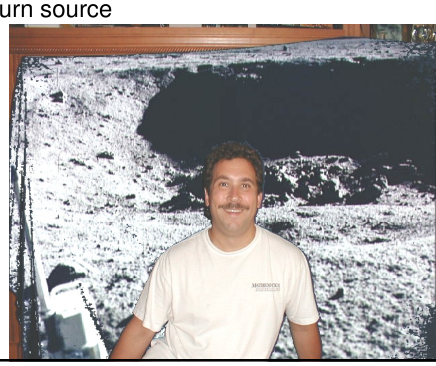


```
def greyscale(picture):
  for p in getPixels(picture):
    redness=getRed(p)
    greenness=getGreen(p)
    blueness=getBlue(p)
    luminance=(redness+blueness+greenness)/3
    setColor(p,
        makeColor(luminance,luminance,luminance))
```

```
def negative(picture):
  for px in getPixels(picture):
    red=getRed(px)
    green=getGreen(px)
    blue=getBlue(px)
    negColor=makeColor(255-red,255-green,255-blue)
    setColor(px,negColor)
```



```
def chromakey2(source,bg):
    for p in getPixels(source):
        if (getRed(p)+getGreen(p) < getBlue(p)):
            setColor(p,getColor(getPixel(bg,getX(p),getY(p))))
    return source
```

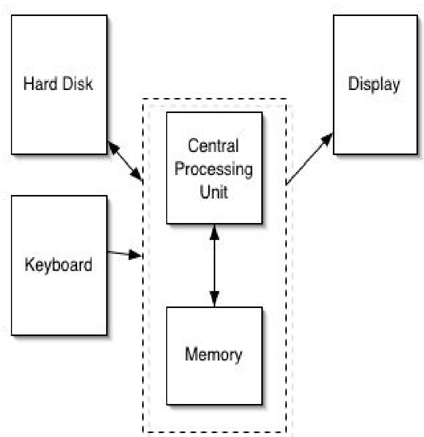


Introduction and Brief History of Programming

- Hardware
  - Physical components that make up a computer
- Computer program or software
  - A self-contained set of instructions used to operate a computer to produce a specific result

How a computer works

- The part that does the adding and comparing is the *Central Processing Unit (CPU)*.
- The CPU talks to the *memory*
  - Think of it as a sequence of millions of mailboxes, each one byte in size, each of which has a numeric *address*
- The *hard disk* provides 10 times or more storage than in memory (20 billion bytes versus 128 million bytes), but is millions of times slower
- The display is the monitor or LCD (or whatever)



Knowing About: Computer Hardware

- Computer hardware components
  - Memory unit
    - Stores information in a logically consistent format
      - Each memory location has an address and data that can be stored there, imagine a long line of mailboxes starting at address 0 and going up to addresses in the billions
    - Two types of memory: RAM and ROM
      - Random Access Memory, Read Only Memory (misnamed)
  - Central Processing Unit
    - Directs and monitors the overall operation of the computer
    - Performs addition, subtraction, other logical operations

Knowing About: Computer Hardware (Continued)

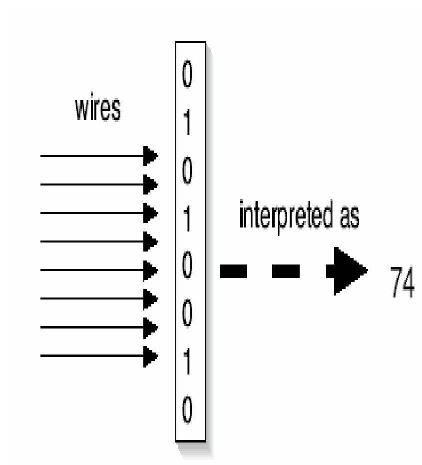
- Evolution of hardware
  - 1950s: all hardware units were built using relays and vacuum tubes
  - 1960s: introduction of transistors
  - mid-1960s: introduction of integrated circuits (ICs)
  - Present computers: use of microprocessors

What computers understand

- It's not really *multimedia* at all.
  - It's *unimedia* (Nicholas Negroponte)
  - *Everything* is 0's and 1's
- Computers are *exceedingly* stupid
  - The only *data* they understand is 0's and 1's
  - They can only do the most simple things with those 0's and 1's
    - Move this value here
    - Add, multiply, subtract, divide these values
    - Compare these values, and if one is less than the other, go follow this step rather than that one.

Key Concept: Encodings

- But we can *interpret* these numbers any way we want.
  - We can *encode* information in those numbers
- Even the notion that the computer understands numbers is an interpretation
  - We encode the voltages on wires as 0's and 1's, eight of these defining a *byte*
  - Which we can, in turn, interpret as a decimal number



Layer the encodings as deep as you want

- One encoding, ASCII, defines an “A” as 65
  - If there’s a byte with a 65 in it, and we decide that it’s a string, POOF! It’s an “A”!
- We can string together lots of these numbers together to make usable text
  - “77, 97, 114, 107” is “Mark”
  - “60, 97, 32, 104, 114, 101, 102, 61” is “<a href=“ (HTML)

What do we mean by *layered* encodings?

- A number is just a number is just a number
- If you have to treat it as a letter, there’s a piece of software that does it
  - For example, that associates 65 with the graphical representation for “A”
- If you have to treat it as part of an HTML document, there’s a piece of software that does it
  - That understands that “<A HREF=“ is the beginning of a link
- That part that knows HTML communicates with the part that knows that 65 is an “A”

Multimedia is unimedia

- But that same byte with a 65 in it might be interpreted as…
  - A very small piece of sound (e.g., 1/44100-th of a second)
  - The amount of redness in a single dot in a larger picture
  - The amount of redness in a single dot in a larger picture which is a single frame in a full-length motion picture

Software (recipes) defines and manipulates encodings

- Computer programs manage all these layers
  - How do you decide what a number should mean, and how you should organize your numbers to represent all the data you want?
  - That's data structures
- If that sounds like a lot of data, it is
  - To represent all the dots on your screen probably takes more than 3,145,728 bytes
  - Each second of sound on a CD takes 44,100 bytes!!

Let's Hear It for Moore's Law!

- Gordon Moore, one of the founders of Intel, made the claim that (essentially) computer power doubles for the same dollar every 18 months.
- This has held true for over 30 years
  - But soon we may be reaching limitations imposed by physics
- Go ahead! Make your computer do the same thing to every one of 3 million dots on your screen. It doesn't care! And it won't take much time either!

First-Generation and Second-Generation (Low-Level) Languages

- Low-level languages
  - First-generation and second-generation languages
  - Machine-dependent languages
  - The underlying representation the machine actually understands
- First-generation languages
  - Also referred to as machine languages
  - Consist of a sequence of instructions represented as binary numbers
  - E.g.: Code to ADD might be 1001 . To add 1+0 and then 1+1 our program might look like this:
    - 1001 0001 0000
    - 1001 0001 0001

First-Generation and Second-Generation (Low-Level) Languages (Continued)

- Second-generation languages
  - Also referred to as assembly languages
  - Abbreviated words are used to indicate operations
  - Allow the use of decimal numbers and labels to indicate the location of the data
- Assemblers
  - Programs that translate assembly language programs into machine language programs
  - Our add program now looks like:
    - ADD 1,0
    - ADD 1,1

Assembler

1001
0001
0000
1001
0001
0001

Third-Generation and Fourth-Generation (High-Level) Languages

- High-level languages
  - Third-generation and fourth-generation languages
  - Programs can be translated to run on a variety of computer types
- Third-generation languages
  - Procedure-oriented languages
  - Object-oriented languages
- Our Add program might now look like:

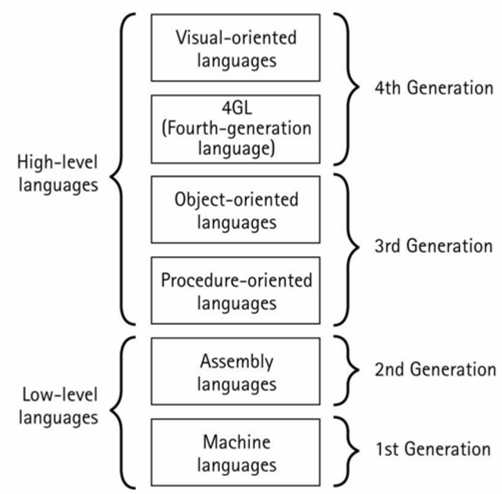
sum = value1 + value2

Compiler

1001
0001
0000
1001
0001
0001

Third-Generation and Fourth-Generation (High-Level) Languages (Continued)

The Evolution of Programming Languages



Computer Science and Media?

- What is computer science about?
- What computers *really* understand
- Media Computation: Why digitize media?
  - How can it possibly work?
- It's about communications and process

What computation is good for

- Computer science is the study of recipes
- Computer scientists study…
  - How the recipes are written (algorithms, software engineering)
  - The units used in the recipes (data structures, databases)
  - What can recipes be written for (systems, intelligent systems, theory)
  - How well the recipes work (human-computer interfaces)

Specialized Recipes

- Some people specialize in crepes or barbeque
- Computer scientists can also specialize on special kinds of recipes
  - Recipes that create pictures, sounds, movies, animations (graphics, computer music)
- Still others look at *emergent properties* of computer "recipes"
  - What happens when lots of recipes talk to one another (networking, non-linear systems)
  - Computer programs to study or simulate natural systems

Key concept: The *COMPUTER* does the recipe!

- Make it as hard, tedious, complex as you want!
- Crank through a million genomes? No problem!
- Find one person in a 30,000 person campus? Sure.
- Process a million dots on the screen or a bazillion sound samples?
  - That's media computation

- Later on we'll see some problems that are computationally too expensive to solve even for the fastest computer today

Why digitize media?

- Digitizing media is encoding media into numbers
  - Real media is *analogue* (continuous)
    - Images
    - Sound
  - To digitize it, we break it into parts where we can't perceive the parts.
- By converting them, we can more easily manipulate them, store them, transmit them without error, etc.

How can it work to digitize media?

- Why does it work that we can break media into pieces and we don't perceive the breaks?
- We can only do it because human perception is limited.
  - We don't see the dots in the pictures, or the gaps in the sounds.
- We can make this happen because we know about *physics* (science of the physical world) and *psychophysics* (psychology of how we perceive the physical world)

Why should you study "recipes"?

- To understand better the recipe-way of thinking
  - It's influencing everything, from computational science to bioinformatics
  - Eventually, it's going to become part of everyone's notion of a liberal education
  - That's the *process* argument
  - BTW, to work with and manage computer scientists
- AND…to communicate!
  - Writers, marketers, producers communicate through computation
- We'll take these in opposite order

Computation for Communication

- All media are going digital
- Digital media are manipulated with software
- You are limited in your communication by what your software allows
  - What if you want to say something that Microsoft or Adobe or Apple doesn't *let* you say?

Programming is a communications skill

- If you want to say something that your tools don't allow, program it yourself
- If you want to understand what your tools can or cannot do, you need to understand what the programs are doing
- If you care about preparing media for the Web, for marketing, for print, for broadcast… then it's worth your while to understand how the media are and can be manipulated.
- Knowledge is Power,
  Knowing how media work is powerful and freeing

We're not going to replace PhotoShop

- Nor ProAudio Tools, ImageMagick and the GIMP, and Java and Visual Basic
- But if you know what these things are doing, you have something that can help you learn new tools

- You are also learning general programming skills that can be applied to creating business applications, scientific applications, etc.
  - Our domain for this class just happens to be (primarily) media

Knowing about programming is knowing about process

- Alan Perlis
  - One of the founders of computer science
  - Argued in 1961 that Computer Science should be part of a liberal education: *Everyone* should learn to program.
    - Perhaps computing is more critical to a liberal education than Calculus
    - Calculus is about rates, and that's important to many.
    - Computer science is about process, and that's important to *everyone*.

A Recipe is a Statement of Process

- A recipe defines how something is done
  - In a *programming language* that defines how the recipe is written
- When you learn the recipe that implements a Photoshop filter, you learn how Photoshop does what it does.
- And that is powerful.

Finally: Programming is about Communicating Process

- A program is the most concise statement possible to communicate a process
  - That's why it's important to scientists and others who want to specify *how* to do something understandably in the most precise words as possible

Python

- The programming language we will be using is called *Python*
  - http://www.python.org
  - Python was invented by researchers across the Internet
  - Considered by many to be the best language to teach programming to beginners, but still powerful enough for real applications
  - It's used by companies like Google, Industrial Light & Magic, Nextel, and others
- The *kind* of Python we're using is called Jython
  - It's Java-based Python
    - More on Java later
  - http://www.jython.org
- We'll be using a specific tool to make Python programming easier, called JES.
  - Invented by the authors of the textbook