Speed

CS A109

Big speed differences

• Many of the techniques we’ve learned take no time at all in other applications
• Select a figure in Word.
  – It’s automatically inverted as fast as you can highlight it.
• Color changes in Photoshop happen as you change the slider
  – Increase or decrease red? Move it and see it happen just as fast as you can move the slider.
Where does the speed go?

• Is it that Photoshop is so fast?
• Or that Jython is so slow?
• It’s some of both—it’s not a simple problem with an obvious answer.
• Let's consider an issue:
  – How fast can computers get?

What a computer *really* understands

• Computers really do not understand Python, nor Java, nor any other language.
• The basic computer only understands one kind of language: *machine language*.
  – instructions to the computer expressed in terms of values in bytes
  – tell the computer to do *very* low-level activities

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
Assembler and machine language

- Machine language looks just like a bunch of numbers.
- *Assembler language* is a set of words that corresponds to the machine language.
  - It’s a one-to-one relationship.
  - A word of assembler equals one machine language instruction, typically.
    - (Often, just a single byte.)

Each *kind* of processor has its own machine language

- Apple computers (used to) use CPU chips called G4 or G5
- Computers running Microsoft Windows may use Pentium processors.
- There are other processors called Alpha, LSI-11, and on and on.

Each processor understands only its *own* machine language
Assembler instructions

• Assembler instructions tell the computer to do things like:
  – Load numbers particular memory locations into special locations (variables) in the computer
    • These special locations are called registers
  – Store numbers into particular memory locations or into special locations (variables) in the computer.
  – Test numbers for equality, greater-than, or less-than.
  – Add numbers together, or subtract them.

An example assembly language program

LOAD #10,R0 ; Load special variable R0 with 10
LOAD #12,R1 ; Load special variable R1 with 12
SUM R0,R1 ; Add special variables R0 and R1
STOR R1,#45 ; Store the result into memory location #45

Recall that we talked about memory as a long series of mailboxes in a mailroom.
Each one has a number (like #45).
The above is equivalent to Python’s: \[ b = 10 + 12 \]
Assembler -> Machine

LOAD 10,R0 ; Load special variable R0 with 10
LOAD 12,R1 ; Load special variable R1 with 12
SUM R0,R1 ; Add special variables R0 and R1
STOR R1,#45 ; Store the result into memory location #45

Might appear in memory as just 12 bytes:
01 00 10
01 01 12
02 00 01
03 01 45

Another Example

• LOAD R1,#65536 ; Get a character from keyboard
• TEST R1,#13 ; Is it an ASCII 13 (Enter)?
• JUMPTRUE #32768 ; If true, go to another part of the program
• CALL #16384 ; If false, call func. to process the new line

Machine Language:
05 01 255 255
10 01 13
20 127 255
122 63 255
Devices are (often) also just memory

- A computer can interact with external devices (like displays, microphones, and speakers) in lots of ways.
- Easiest way to understand it (and is often the actual way it’s implemented) is to think about external devices as corresponding to a memory location.
  - Store a 255 into memory location 65542, and suddenly the red component of the pixel at (101,345) on your screen is set to maximum intensity.
  - Everytime the computer reads memory location 897784, it’s a new sample just read from the microphone.
- So the simple loads and stores handle multimedia, too.

Machine language is executed very quickly

- A mid-range laptop these days has a clock rate of 1.5 Gigahertz.
- What that means exactly is hard to explain, but let’s interpret it as processing 1.5 billion bytes per second.
- Those 12 bytes would execute inside the computer, then, in 12/1,500,000,000th of a second!
Applications are typically compiled

• Applications like Adobe Photoshop and Microsoft Word are compiled.
  – This means that they execute in the computer as pure machine language.
  – They execute at that level speed.
• However, Python, Java, Scheme, and many other languages are (in many cases) interpreted.
  – They execute at a slower speed.
  – Why? It’s the difference between translating instructions and directly executing instructions.

An example

• Sample Problem:
  Write a function doGraphics that will take a list as input. The function doGraphics will start by creating a canvas from the 640x480.jpg file in the mediasources folder. You will draw on the canvas according to the commands in the input list.
  
  Each element of the list will be a string. There will be two kinds of strings in the list:
  
  • "b 200 120" means to draw a black dot at x position 200 y position 120. The numbers, of course, will change, but the command will always be a "b". You can assume that the input numbers will always have three digits.
  • "l 000 010 100 200" means to draw a line from position (0,10) to position (100,200)
  
  So an input list might look like: ["b 100 200","b 101 200","b 102 200","l 102 200 102 300"] (but have any number of elements).
Sample Solution

def doGraphics(mylist):
    canvas =
        makePicture(getMediaPath("640x480.jpg"))
    for i in mylist:
        if i[0] == "b":
            x = int(i[2:5])
            y = int(i[6:9])
            print "Drawing pixel at ",x,";",y
            setColor(getPixel(canvas, x,y), black)
        if i[0] == "l":
            x1 = int(i[2:5])
            y1 = int(i[6:9])
            x2 = int(i[10:13])
            y2 = int(i[14:17])
            print "Drawing line at",x1,y1,x2,y2
            addLine(canvas, x1, y1, x2, y2)
    return canvas

This program processes each string in the command list.

If the first character is "b", then the x and y are pulled out, and a pixel is set to black.

If the first character is "l", then the two coordinates are pulled out, and the line is drawn.

Running doGraphics()

>>> canvas=doGraphics("
b 100 200","b 101 200","b 102 200","l 102 200 102 300","l 102 300 200 300")
Drawing pixel at  100 : 200
Drawing pixel at  101 : 200
Drawing pixel at  102 : 200
Drawing line at 102 200 102 300
Drawing line at 102 300 200 300
>>> show(canvas)
We’ve invented a new language

• ["b 100 200","b 101 200","b 102 200","l 102 200 102 300","l 102 300 200 300"] is a program in a new graphics programming language.

• Postscript, PDF, Flash, and AutoCAD are not too dissimilar from this.
  – There’s a language that, when interpreted, “draws” the page, or the Flash animation, or the CAD drawing.

• But it’s a slow language!

Would this run faster?
Does the exact same thing

```python
def doGraphics():
    canvas =
        makePicture(getMediaPath("640x480.jpg"))
    setColor(getPixel(canvas,
        100,200),black)
    setColor(getPixel(canvas,
        101,200),black)
    setColor(getPixel(canvas,
        102,200),black)
    addLine(canvas, 102,200,102,300)
    addLine(canvas, 102,300,200,300)
    show(canvas)
    return canvas
```
Which do you think will run faster?

```python
def doGraphics(mylist):
    canvas = makePicture(getMediaPath("640x480.jpg"))
    for i in mylist:
        if i[0] == "b":
            x = int(i[2:5])
            y = int(i[6:9])
            print "Drawing pixel at ",x, ":",y
            setColor(getPixel(canvas, x,y),black)
        if i[0] == "l":
            x1 = int(i[2:5])
            y1 = int(i[6:9])
            x2 = int(i[10:13])
            y2 = int(i[14:17])
            print "Drawing line at",x1,y1,x2,y2
            addLine(canvas, x1, y1, x2, y2)
    return canvas
```

def doGraphics():
    canvas = makePicture(getMediaPath("640x480.jpg"))
    setColor(getPixel(canvas, 100,200),black)
    setColor(getPixel(canvas, 101,200),black)
    setColor(getPixel(canvas, 102,200),black)
    addLine(canvas, 102,200,102,300)
    addLine(canvas, 102,300,200,300)
    show(canvas)
    return canvas

Above just draws the picture.
The left one figures out (interprets) the picture, then draws it.

Could we *generate* that second program?

- What if we could write a function that:
  - Takes ["b 100 200", "b 101 200","b 102 200","l 102 200 102 300","l 102 300 200 300"]
  - Writes a file that is the Python version of that program.

```python
def doGraphics():
    canvas = makePicture(getMediaPath("640x480.jpg"))
    setColor(getPixel(canvas, 100,200),black)
    setColor(getPixel(canvas, 101,200),black)
    setColor(getPixel(canvas, 102,200),black)
    addLine(canvas, 102,200,102,300)
    addLine(canvas, 102,300,200,300)
    show(canvas)
    return canvas
```
Introducing a *compiler*

```python
def makeGraphics(mylist):
    file = open("graphics.py", "wt")
    file.write("def doGraphics():\n")
    file.write("  canvas = makePicture(getMediaPath("640x480.jpg"))\n")
    for i in mylist:
        if i[0] == "b":
            x = int(i[2:5])
            y = int(i[6:9])
            print "Drawing pixel at ",x,"",y
            file.write("  setColor(getPixel(canvas, "+str(x)+","+str(y)+"),black)\n")
        if i[0] == "l":
            x1 = int(i[2:5])
            y1 = int(i[6:9])
            x2 = int(i[10:13])
            y2 = int(i[14:17])
            print "Drawing line at",x1,y1,x2,y2
            file.write("  addLine(canvas, "+str(x1)+","+str(y1)+","+str(x2)+","+str(y2)+")\n")
            file.write("  show(canvas)\n")
    file.write("  return canvas\n")
    file.close()
```

Compilers are amazing

- Think about what that last program does:
  - It inputs a program in one language (our mini l/b graphics language)
  - And *generates* another program in another language (Python) that does the same thing.
- It’s a program that writes programs!
  - Given a specification of a process.
  - Create a specification of the same process, but in another notation.
Why do we write programs?

• One reason we write programs is to be able to do the same thing over-and-over again, without having to rehash the same steps in Photoshop each time.
• A compiler makes that re-running the program a thousand times faster.

Which one leads to shorter time overall?

• Interpreted version:
  – 100 times
    • doGraphics(["b 100 200","b 101 200","b 102 200","l 102 200 102 300","l 102 300 200 300"]) involving interpretation and drawing each time.

• Compiled version
  – 1 time makeGraphics(["b 100 200","b 101 200","b 102 200","l 102 200 102 300","l 102 300 200 300"])
    • Takes as much time (or more) as interpreting.
    • But only once
  – 100 times running the very small graphics program.
Applications are compiled

• Applications like Photoshop and Word are written in languages like C or C++
  – These languages are then compiled down to machine language.
  – That stuff that executes at a rate of 1.5 billion bytes per second.
• Jython programs are interpreted.
  – Actually, they’re interpreted twice!

Java programs typically don’t compile to machine language.

• Recall that every processor has its own machine language.
  – How, then, can you create a program that runs on any computer?
• The people who invented Java also invented a make-believe processor—a virtual machine.
  – It doesn’t exist anywhere.
  – Java compiles to run on the virtual machine
    • The Java Virtual Machine (JVM)
What good is it to run only on a computer that doesn’t exist?!?

• Machine language is a very simple language.
• A program that *interprets* the machine language of some computer is not hard to write.

```python
def VMinterpret(program):
    for instruction in program:
        if instruction == 1:  # It's a load
            ...
        if instruction == 2:  # It's an add
            ...
```

Java runs on everything…

• Everything that has a JVM on it!
• Each computer that can execute Java has an *interpreter* for the Java machine language.
  – That interpreter is usually compiled to machine language, so it’s very fast.
• Interpreting Java machine is pretty easy
  – Takes only a small program
• Devices as small as wristwatches can run Java VM interpreters.
Running a Java Program

```
Public class Foo {
    if (e.target=xyz) then
        this.hide();
    }
```

Java Program

Java compiler

```
01010001 01010010 01010010
```

Java Byte Code – Would run on The Java Virtual Machine

Mac Interpreter

PC Interpreter

PalmPilot Interpreter

What happens when you execute a Python statement in JES

- Your statement (like “show(canvas)”) is first compiled to Java!
  - Really! You’re actually running Java, even though you wrote Python!
- Then, the Java is compiled into Java virtual machine language.
  - Sometimes appears as a .class or .jar file.
- Then, the virtual machine language is interpreted by the JVM program.
  - Which executes as a machine language program (a .exe)
Is it any wonder that Python programs in JES are slower?

• Photoshop and Word simply execute.
  – At 1.5 Ghz and faster!
• Python programs in JES are compiled, then compiled, then interpreted.
  – Three layers of software before you get down to the real speed of the computer!
• It only works at all because 1.5 billion is a REALLY big number!

Why interpret?

• For us, to have a command area.
  – Compiled languages don’t typically have a command area where you can print things and try out functions.
  – Interpreted languages help the learner figure out what’s going on.
• For others, to maintain portability.
  – Java can be compiled to machine language.
    • In fact, some VMs will actually compile the virtual machine language for you while running—no special compilation needed.
  – But once you do that, the result can only run on one kind of computer.
  – Programs for Java (.jar files typically) can be moved from any kind of computer to any other kind of computer and just work.
  – Also good for many web apps, since speed often not a key requirement