What did you think of Lightbot?
As Experienced Lightbot Hackers

- What are you doing in Lightbot?
  - Commanding a robot through a “blocks world”
  - Programming is **commanding** an agent
Other aspects of “commanding”

- The agent is usually a computer, but it could be a person, or other device (e.g. a robot)
- The agent follows the commands, the instructions, without making a mistake, and without stopping, doing only what it is asked
- The program implements your intentions – it does what you tell it to do
- You try to get the robot to the Blue Tile goal – that’s the point of the instructions that you gave it
Sequencing

- Instructions are *given* in sequence
- They are *executed* in sequence – essential
  - Instructions ...
    - From a limited repertoire
    - All are within agent’s ability; no JUMP_3
    - Executed one-at-a-time
  - A “program counter” keeps track of agent’s progress
Instructions Formed of Simpler Instructions

- Check out this screen shot of the Lightbot
- It is partway through an instruction ... its beacon is lit, but not the tile
- To a programmer the instruction is monolithic (one thing)
- To an agent each instruction is a series of steps

An Instruction *abstracts* those steps
Abstraction

- The word “abstraction” is used a lot in computing
- Remember: it was one of the 7 big ideas
- *Abstraction is a way to understand and solve problems*
- As a general definition, abstraction *eliminates details to focus on essential properties*
- The instruction example just given illustrates *functional abstraction* meaning that we have given a *name* to a *series of operations* that perform a coherent (and to us meaningful) activity; the name is the instruction, the series of operations are the bot’s actions to implement it
Collecting the operations together and giving them a name is *functional abstraction*

- The group of operations perform some function but we ignore all of the details
- Giving it a name is *functional* abstraction
- This is AMAZINGLY powerful
- What makes it powerful, is we can forget about the operations and think only about the function they do; more about this later

Let’s do some functional abstraction
Example: Abstraction in Everyday life

- Get Dressed
  - Dress Bottom Half
    - Put on Sox
  - Dress Top Half
    - Put on Pants
    - Put on Shirt
Functions abstract by packaging Computation

- $F_1()$ packages actions: E.G. “process a riser”
Functions abstract by packaging Computation

- Here the “main” method has just 9 commands.
The Function Becomes A Concept

- Because $F_1(\cdot)$ “processes a riser,” I think of the programming task as

<table>
<thead>
<tr>
<th>Process a riser</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move to next riser</td>
<td>F1</td>
</tr>
<tr>
<td>Process a riser</td>
<td>F1</td>
</tr>
<tr>
<td>Move to next riser</td>
<td>F1</td>
</tr>
<tr>
<td>Process a riser</td>
<td>F1</td>
</tr>
</tbody>
</table>

- With $F_1(\cdot)$ as a concept, I simplify the programming to just 9 steps rather than 21 OR

- It also suggests another concept:
  - $\text{Move\_to\_next\_riser}(\cdot)$
A Five Instruction Program: Define F1, F2, interleave

New way of thinking about the problem
Here Is What Is Beautiful …

- Did everyone see 1 idea, 2 applications?

Slide 5

- To a programmer the instruction is monolithic (one thing)
- To an agent each instruction is a series of steps

Slide 12

F1( ): Process Riser
F2( ): Move To Next Riser

It is one concept here (monolithic, but here it is a series of eight instructions
Abstraction …

- Formulating blocks of computation as a “concept” is functional abstraction
- What we did is important and we do it all the time in everyday life (otherwise we couldn’t deal with life’s complexity)...
  - We break a task down into (one or more) subtasks
  - We solve a subtask using a sequence of instructions
  - We put the solution into a function “package”, gave it a name, “process a riser,” and thus created a new thing, a concept, something we can talk about & use
  - Then we used it to solve something more complicated … and probably repeat this approach at the next higher level
- This lets us do more complicated things
True or false: Functional abstraction allows an agent (the robot for example) to complete a task in fewer steps.
Recursion: The repeated application of a recursive procedure or definition.
Recursive examples

A string of digits is a digit followed by a string of digits.
First Try: Recursion 2
Second Try: Recursion 2. Just keeps going....
Third Try: Recursion 2: Finds stop condition!
Programming Rule 1: Try, try again!!

It only does what you tell it to do
Recursion 3!: First Try
Recursion 3!: First Try
Recursion 3: OOPS!! Second Try
Recursion 3: Third Try...... Wow!
Recursion 4: Whoa!!! What is F2 subroutine?
What do you think?
Recursion 4: Whoa!!! What is F2 subroutine?
Recursion 4: We keep trying
Move two and turn around?

Would this function ever be useful?

A. Yes
B. No
Another way to use a function for the risers
Summary From Lightbot 2.0

- Programming is **commanding** an agent
  - **Agent**: usually a computer, person, or other device
  - Agent follows **instructions**, flawlessly & stolidly
  - The program implements human intent
- Instructions are *given* in sequence
- ... and *executed* in sequence
  - Limited repertoire, within ability, one-at-a-time
  - “Program counter” keeps track current instruction
- Formulating computation as a “concept” is **functional abstraction**
Summary: Abstraction

- An instruction (of the Lightbot or any other computer) is **abstracted** into the command name; functions **abstract** useful and meaningful operations, functions abstract functions built of functions, etc.

- Layer upon layer, we build software solutions
Homework 2: Lightbot in Symbolic Form

Goal: The purpose of this assignment is to look at textual ways of programming the bot.

When we solved the Lightbot 2.0 exercises last time, we programmed the bot using a tiny list of instructions presented iconographically, that is, as pictures. But, these could just as easily be written symbolically, that is, as text. So, our instruction list

Step Right Left Jump Power F.name

can be expressed symbolically,

as shown. When we program symbolically, we don’t have to call the functions F1 and F2. We can give them actual names like F.turn_around for a function that causes the bot to turn around. So, we can give the symbolic solution to
“Digits” does not refer only to your 10 fingers…

Digitization & Processing

Used with permission from
Lawrence Snyder
University of Washington, Seattle
A Short History of Digital Info

- One goal of CS Principles is to understand how computers and digital information are “game changers,” how they create opportunities
- I will do that by highlighting progress of “data processing” over last 120 years or so (it’s very incomplete)
  - Digitization, computers, ICs, transistors, PCs, Internet, and WWW are key
  - Focus on advances since ...
Only **people** can read it ... [Though recently, *some* progress in handwriting analysis has occurred; limited use.]

- First serious advance in digitization: punch cards
- Herman Hollerith develops idea for 1890 census

Hollerith Card, Courtesy IBM
Machines Process Digital Data

- Mechanical methods – sensing a hole in a card or not – allows machines to help w/work
No Computer Needed To Process Data

- A mechanical machine can “read” a card with a “metal brush”
Sensing Punch Allows Some Action

- When the circuit closes, some mechanical action can happen
Suppose Hollerith coded men as 0, women as 1.

How many men and women in the population?

Machine Reads Cards, Puts women in this slot
Puts men in this slot
... producing 2 piles
Run each pile through again just to count them -- done
Meanwhile, w/o Digital Data

- Poor Kermit must go through census sheets, counting (and probably making mistakes)

The message: “Digitizing” makes information discrete, it’s either there (1) or not (0), and a machine can determine that fact using mechanical or electronic means. Once data is digital, it is just a matter for engineers to build more capable machines.
After processing based on reading cards, a machine can “save its work” by punching cards.
Next Big Things ... Very Big!

- Electronic computers came after WWII
By Mid 20\textsuperscript{th} Century ~ 1960

- Large and medium-size companies used card based digital data; mechanical processing
- Computers began to replace mechanical b/c a computer’s “processing instructions” (program) could be easily changed, & they perform more complex operations – flexibility
- Computers, memory much more expensive –

Message: Computers take the task specification (program) and digital data as inputs, making them very versatile machines; one machine does it all! Programming becomes critical technology.
Key to Modern “Computers”

The key difference between the early tabulating machines and modern computers is
A. mechanical vs electrical
B. non-programmable vs programmable
C. decimal vs binary
D. all of the above
Next Big Things: Integrated Circuits

- Transistors – solid state switching
- Integrated Circuit – all circuit parts fabbed at once from similar materials
A transistor is a switch: If the gate (black bar) is neutral, charge cannot pass; if gate is charged, the wires are connected.
Transistors are smart, but “wiring them up” with other parts was labor intensive

**Integrated circuits** – transistors + resistors + capacitors – are created together in one long recipe – small, cheap, reliable

Key fabrication process is *photolithography* – the transistors are “printed” on the silicon!
Photolithography
Integrated Circuits

Message: Transistors switch wires on and off in solid material (no moving parts to wear out) and ICs are fabbed as a unit (no wiring) and using photolithography – complexity of circuit doesn’t matter! We can all have a computer.
A transistor is most like
A. A hole (or lack of) in a punched card
B. A light switch
C. An entire punched card with multiple holes
D. A whole bank of light switches
Next Big Thing: Personal Computers

- Ken Olsen, Founder of Digital Equipment, “There is no reason for any individual to have a computer in their home [1977]”

© 2010 Larry Snyder, CSE
Computing Comes To Everyone

- Regular folks – not just government, military, scientists, banks and companies – could now apply computers to their interests
- Created a demand for digital data: news, pics, audio, video, books, etc., causing old technologies to digitize rapidly. Now it matters to everyone if a machine can “read” it
- From about 1985 most “new” information has been digital
- Quickly, people acquired enormous amounts of information
Message: Computers can be easily transformed to do new things, and being cheap, we can all have some, motivating us to want digital everything.
Next Big Thing: Internet

- Invented in 1969, it took almost 20 years to get out of the lab and into public consciousness.

“On the Internet, nobody knows you’re a dog.”
Wikipedia Timeline

en.wikipedia.org/wiki/Timeline_of_computing
Connecting Up

- Computers are useful; connected computers are awesome
- If \( n \) computers are connected, adding one more gives \( n \) new connections!
- Communication with friends or businesses all over the world became easy and casual – some people even found out about time zones
- Digital media allows people to share each other’s information at no cost
Message: The Internet is a general mechanism to communicate digital data – it doesn’t matter what it is: music, email, video …
Today, all computers “speak” a common language: hyper-text transfer protocol
Two phenomena make the WWW brilliant

- All computers use one standard protocol (http) meaning for once all of the world’s people – who don’t speak one language – have a surrogate that does

- Publishing and accessing information is completely decentralized – generally, no one limits what you put out or go after
Seeing Other People’s Digital Info

Message: WWW exploits one protocol, neutralizing differences at endpoints; the Internet’s universal medium lets us look at other people’s digital info.
Key principle of digital encoding: Physically, information is the presence or absence of a phenomenon at a given place and time!

Card example:
- Phenomenon – hole in the card
- Present – detected by brush making elec contact
- Absent – brush insulated from electrical source
- Place – there are several on the card; devices can know the positions
- Time – hole is permanent representation of info
A General Idea

- Digital Information: Detecting the presence or absence of a phenomenon at a specific place and time: PandA
- Phenomena: light, magnetism, charge, mass, color, current, ...
- Detecting depends on phenomenon – but the result must be discrete: it was detected or not; there is no option for “sorta there”
- Place and time apply, but usually default to “obvious” values; not so important to us
Digital Discussion

- Alternatives to detecting the hole in a card
Digital Discussion

- Alternatives to detecting the hole in a card
- Sidewalk Memory – squares and rocks
Digital Discussion

- Alternatives to detecting the hole in a card
- Sidewalk Memory – squares and rocks

Other phenomena … CD ROM how it works:
PandA is a *binary representation* because it uses 2 patterns.

Bit -- it’s a contraction for “binary digit”

-- a position in space/time capable of being set and detected in 2 patterns
A byte is eight bits treated as a unit
- Adopted by IBM in 1960s
- A standard measure ever since
- Bytes encode the Latin alphabet using ASCII -- the American Standard Code for Information Interchange
### ASCII

<table>
<thead>
<tr>
<th>ASCII</th>
<th>0000</th>
<th>0001</th>
<th>0010</th>
<th>0011</th>
<th>0100</th>
<th>0101</th>
<th>0110</th>
<th>0111</th>
<th>1000</th>
<th>1001</th>
<th>1010</th>
<th>1011</th>
<th>1100</th>
<th>1101</th>
<th>1110</th>
<th>1111</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N_1</td>
<td>S_1</td>
<td>E_1</td>
<td>E_2</td>
<td>A_1</td>
<td>B_1</td>
<td>B_2</td>
<td>H_1</td>
<td>L_1</td>
<td>Y_1</td>
<td>F_1</td>
<td>C_1</td>
<td>S_5</td>
<td>S_6</td>
<td>S_7</td>
<td>S_8</td>
</tr>
<tr>
<td></td>
<td>D_1</td>
<td>D_2</td>
<td>D_3</td>
<td>D_4</td>
<td>N_2</td>
<td>E_3</td>
<td>C_2</td>
<td>E_4</td>
<td>S_9</td>
<td>E_5</td>
<td>S_10</td>
<td>E_6</td>
<td>S_11</td>
<td>S_12</td>
<td>S_13</td>
<td>S_14</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>!</td>
<td>&quot;</td>
<td>#</td>
<td>$</td>
<td>%</td>
<td>&amp;</td>
<td>'</td>
<td>(</td>
<td>)</td>
<td>*</td>
<td>+</td>
<td>,</td>
<td>-</td>
<td>.</td>
<td>/</td>
<td>&lt;</td>
</tr>
<tr>
<td></td>
<td>@</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>[</td>
<td>\</td>
<td>]</td>
<td>^</td>
<td>_</td>
</tr>
<tr>
<td></td>
<td>`</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td>o</td>
</tr>
<tr>
<td></td>
<td>p</td>
<td>q</td>
<td>r</td>
<td>s</td>
<td>t</td>
<td>u</td>
<td>v</td>
<td>w</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>{</td>
<td></td>
<td></td>
<td>}</td>
<td>~</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>:</td>
<td>;</td>
<td>&lt;</td>
<td>=</td>
<td>&gt;</td>
<td>?</td>
</tr>
<tr>
<td></td>
<td>8_0</td>
<td>8_1</td>
<td>8_2</td>
<td>8_3</td>
<td>8_4</td>
<td>8_5</td>
<td>8_6</td>
<td>8_7</td>
<td>8_8</td>
<td>8_9</td>
<td>8_A</td>
<td>8_B</td>
<td>8_C</td>
<td>8_D</td>
<td>8_E</td>
<td>8_F</td>
</tr>
<tr>
<td></td>
<td>9_0</td>
<td>9_1</td>
<td>9_2</td>
<td>9_3</td>
<td>9_4</td>
<td>9_5</td>
<td>9_6</td>
<td>9_7</td>
<td>9_8</td>
<td>9_9</td>
<td>9_A</td>
<td>9_B</td>
<td>9_C</td>
<td>9_D</td>
<td>9_E</td>
<td>9_F</td>
</tr>
<tr>
<td></td>
<td>10_0</td>
<td>10_1</td>
<td>10_2</td>
<td>10_3</td>
<td>10_4</td>
<td>10_5</td>
<td>10_6</td>
<td>10_7</td>
<td>10_8</td>
<td>10_9</td>
<td>10_A</td>
<td>10_B</td>
<td>10_C</td>
<td>10_D</td>
<td>10_E</td>
<td>10_F</td>
</tr>
<tr>
<td></td>
<td>11_0</td>
<td>11_1</td>
<td>11_2</td>
<td>11_3</td>
<td>11_4</td>
<td>11_5</td>
<td>11_6</td>
<td>11_7</td>
<td>11_8</td>
<td>11_9</td>
<td>11_A</td>
<td>11_B</td>
<td>11_C</td>
<td>11_D</td>
<td>11_E</td>
<td>11_F</td>
</tr>
</tbody>
</table>

© 2010 Lawrence Snyder, CSE
With 8 places how many different letters?
UTF-8: All the alphabets in the world

- Uniform Transformation Format: a variable-width encoding that can represent every character in the Unicode Character set
- 1,112,064 of them!!!
- UTF-8 is the dominant character encoding for the World-Wide Web, accounting for more than half of all Web pages.
- The Internet Engineering Task Force (IETF) requires all Internet protocols to identify the encoding used for character data
- The supported character encodings must include UTF-8.
Recap: Next Big Thing

- Electronic Computers
- Integrated Circuits
- Personal Computers
- Internet
- WWW + http
- ???